

Michael J. Wagner

MACHINE LANGUAGE DISK I/O & OTHER MYSTERIES



Machine Language Disk I/O for the TRS-80 models I and III

TRS-80 INFORMATION SERIES - VOLUME V

Michael J. Wagner

MACHINE LANGUAGE DISK I/O & OTHER MYSTERIES

David Moore — Editor

Charles Trapp — Assistant Editor

D. J. Smith — Cover Design and Graphics

First Edition

Second Printing

June, 1983

Printed in the United States of America

Copyright ©1982 by Michael J. Wagner

ISBN 0 936200 06 5

All rights reserved. No Part of this book may be reproduced by any means without the express written permission of the publisher. Example programs are for personal use only. Every reasonable effort has been made to ensure accuracy throughout this book, but neither the author or publisher can assume responsibility for any errors or omissions. No liability is assumed for any direct, or indirect, damages resulting from the use of information contained herein.



Published by

IJG Inc
1953 West
11th Street
Upland, CA
91786 (714)
946-5805

Radio Shack, TRSDOS and TRS-80 are registered trademarks of the Tandy Corporation. LDOS is a registered trademark of Logical Systems, Inc. NEWDOS is a registered trademark of Apparat, Inc., and VTOS is a registered trademark of Virtual Technology, Inc.

IMPORTANT

Read This Notice

Any software or hardware information is used at your own risk. Neither the PUBLISHER nor the AUTHOR assumes any responsibility or liability for loss or damages caused or alleged to be caused, directly or indirectly, by applying any information or alteration to software or hardware described in this book, including but not limited to: any interruption of service, loss of business, anticipatory profits or consequential damages resulting from the use or operation of such hardware or software information or alterations. Also, no patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this book, the PUBLISHER and the AUTHOR assume no responsibility for errors or omissions. The reader is the sole judge of his or her skill and ability to use the information or alterations contained in this book.

Editor's Note

About the Author

Michael Wagner is currently on staff at Softronics. He has 10 years of experience in Radio electronics, and is into Amateur Radio (his call sign is N6FYX). Mike has had what he calls "more than enough" experience on main frame computer systems, and has been suffering from an acute case of 'microcodius bitcrunchius' for the last 4 years. Most of his time is spent writing software for TRS-80 computers. His main form of relaxation is, as he states it, "simply goofing off." He is also a semi-professional rock guitarist-at-large, and enjoys heavy conversation in religion, politics, mathematics, cosmology and *anything* controversial.

I'd like to say a word of thanks to:
'Biffy' of Western Digital, Inc., for the latest editions of the data and applications sheets for the FD1771-01 and the FD179X;
Nancy DeDiemar of Helen's Place, for teaching me the value of patience and being patient with me;
Eric Jorgensen of Clymer Publications, for managing to correct those 'little' mistakes I didn't know I'd made;
Cindy Hall, for her enthusiastic support and having the best lungs in the business;
... and I'd like to express a special kind of thanks for the faith shown by all of you who buy our books before they're released

David E. Moore — Editor
September, 1982

Table of Contents

Preface	8
Chapter 1:	
What is a Mini-Floppy?	9
Tracks and Sectors	11
A Technical Explanation of the Controller	12
Addresses Used in Controlling the FDC	13
The Command/Status Register	15
TRS-80 Model 1 Disk Format	16
Model III Supplement to Chapter 1:	
Model III Disk Controlling System	21
Model III Non-maskable Interrupts	23
Model III Double-Density Disk Format	24
 Chapter 2:	
Selecting a Drive	27
Reading a Selected Drive's Status	28
Hello, Is Anyone Home?	29
Model III Supplement to Chapter 2:	
Selecting Model III Drives	32
Checking the Status of Model III Drives	33
 Chapter 3:	
Head Positioning	35
The RESTORE Command	38
The STEP-IN Command	40
The STEP-OUT Command	42
The STEP Command	44

The SEEK Command	47
The FORCE INTERRUPT Command	49
Model III Supplement to Chapter 3:	
Model III Head Positioning	50
 Chapter 4:	
Disk Data Input/Output Commands	51
Programmed I/O	51
The READ ADDRESS Command	54
The READ TRACK Command	55
The WRITE TRACK or FORMAT Command	57
The READ SECTOR Command	58
The WRITE SECTOR Command	60
Model III Supplement to Chapter 4:	
The NMI — Post I/O Processing	62
The Model III READ ADDRESS Command	63
The Model III READ TRACK Command	64
The Model III WRITE TRACK Command	64
The Model III READ SECTOR Command	64
The Model III WRITE SECTOR Command	64
 Chapter 5:	
DISKIO — Full Sector I/O Routine	65
The Drive Control Table	71
The I/O RE-TRIES Value	72
Explanation of I/O Errors	72
Single-Byte I/O versus Sector I/O	73
Model III Supplement to Chapter 5:	
The DCT	77
Using the Disk Drive in a Program	77
Model III Disk Driver Routine	78
 Chapter 6:	
Using TRSDOS/NEWDOS/LDOS Disk Routines	83
TRSDOS/NEWDOS/VTOS Disk Files	83
Opening a New or Existing File	85
Opening an Existing File	86
Performing Direct Record I/O	87
Performing Single-Byte I/O	88
Closing Files	89
Killing Files	90

File Handling and Error Processing	90
Past EOF Messages	93
Other DOS Functions	94
Other DOS Trivia	95
Model III Supplement to Chapter 6:	
Single-Density Disk Format	98
 Chapter 7:	
TRS-80 Model I Interrupts	101
Model III Supplement to Chapter 7:	
The Model III Interrupt System	105
Handling Model III Interrupts	107
RS-232 Interrupts	108
Port-Mapped Devices and Interrupts	108
Real-time Clock Interrupts	108
The 1500 Baud Cassette Interrupts	109
 Chapter 8:	
Handy Routines, Drivers and Programs	111
TRSDOS Error Displayer	111
Disk Formatter Program	113
PASSFIND, Password Finder	118
LOADER BAS — Load Format Displayer	121
ASCIIZAP — ASCII Sector Modifier	126
 Chapter 9:	
Miscellaneous Junk	135
FDC Quick-Reference Chart of Commands	135
Stepping Speeds of Popular Drives	136
Small Disk Operating System (S/OS)	137
Technical Information Section	137
Disk I/O Primitives	138
S/OS File I/O Vector Calls	140
S/OS Special Function Calls	141
The S/OS User Interface	142
Drive Control Tables	143
The S/OS File Control Block	144
 Appendix I:	
The Best Term Program in This Book	175
Running TERM	175
Explanation of the Menu Functions	176

Appendix II:	
Western Digital Data and Applications	196

Glossary	263
-----------------------	------------



Preface

After getting severely “bitten” by the personal-computer bug a few years ago, and looking at my disk drives for many hours, I finally got the urge to find out how to control them. I love system stuff, — controlling and such. I had this preoccupation to write a disk operating system (to be released as EZ/DOS in the coming months) for the Model I and III. I thought it would be a good experience, but I didn’t know what I was up against. Not that disk I/O is extremely difficult, — it was just hard to find any *complete* documentation on the subject. I had found a few articles that were somewhat helpful, but they lacked complete coverage of the disk I/O subject. After spending many weeks and late nights deciphering disk-controller chip specification sheets and consuming several six packs of root beer, I knew how to completely control the TRS-80 Model I disk system.

This book is the result of my “discoveries” on the subjects of disk I/O and utilizing the Model I’s interrupt system. The purpose of this book is to inform anyone familiar with Z-80 assembly-language programming how to control the TRS-80 Model I and III disk drive interrupt systems. Driver routines for every function described, with abundant examples, are included in this book. It also covers utilization of TRSDOS assembly-language file I/O calls and techniques.

This book was composed and edited using a TRS-80 Model I with 48K of RAM, an LNW 5/8 doubler, an RS-232, two Tandon 40-track drives, an Epson MX-80 printer, a Spinterm printer, Electric Pencil word-processing software, and lots of root beer and late-night radio talk-shows.

I would like to give special thanks to the following people who have helped make this book a reality: Jewell and Joe, Harv Pennington, Western Digital Corp., Tandy Corp., and all the nice folks at IJG.

Mike Wagner

April, 1982

1

What is a Mini-Floppy?

The mini-floppy diskette system used by the TRS-80 Model I and many other microcomputers is a miniaturized version of the 8-inch floppy disk systems used in many microcomputer and minicomputer installations. The diskette itself is a flat, round piece of plastic that is coated with a magnetically sensitive oxide material. It looks similar to a 45 RPM record, but there aren't any grooves on it. The diskette has a 1 1/8-inch centering hole in its middle and an off-center index hole used by the drive for timing the revolutions. The disk is provided with a protective jacket to prevent finger prints or foreign matter from contaminating the diskette's surface. Users are warned (usually on the disk jacket) not to touch the exposed areas of the disk and not to write on the jacket with anything harder than a felt-tip pen. Also, the disks are heat sensitive, so leaving a disk inside an automobile will usually cause destruction of the jacket within an hour.

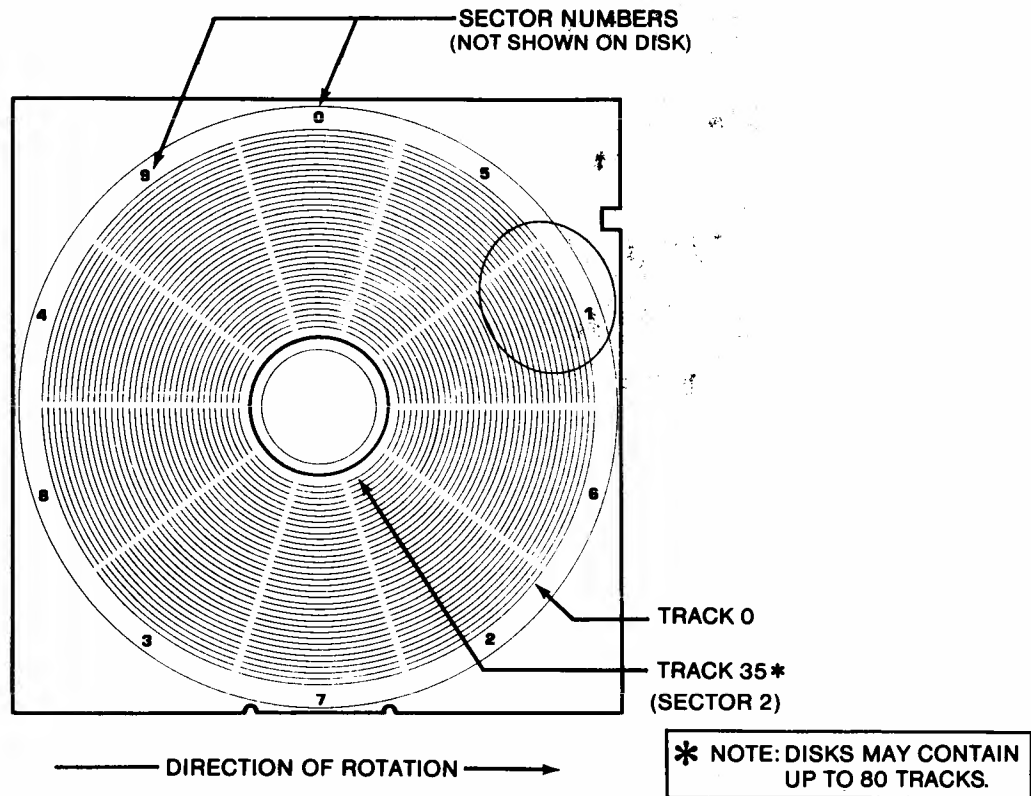
For microprocessor based systems, the 5 1/4-inch mini-floppy comfortably fills the gap between the serial-analog tape units (which are generally too slow for most peoples' patience and somewhat unreliable during high-volume data transfers), the higher cost of 8-inch floppy disk systems, and hard disk systems. Also, they are much faster than paper tape or punch-card systems.

Data is stored on the diskettes by making some parts of the disk more magnetically positive and other parts more magnetically negative. In IBM-compatible 3370 single-density installations (such as the Model I), the data is stored on the disk as one pulse in a given time as logical 0, and two pulses as logical 1. The writing, reading and converting of these pulses into full 8-bit bytes (characters) is the job of the Floppy Disk Controller in your expansion interface, which I shall refer to from now on as simply 'the controller'.

The disk drives themselves are not *too* sophisticated, although they perform some timing functions. The drives basically contain three elements. The first is a motor system that rotates the diskettes inside their protective jackets at a rate of 300 RPM (or 5 rotations per second). Second, the drives contain a read/write head that is used to read and write data to and from the diskettes. This read/write head

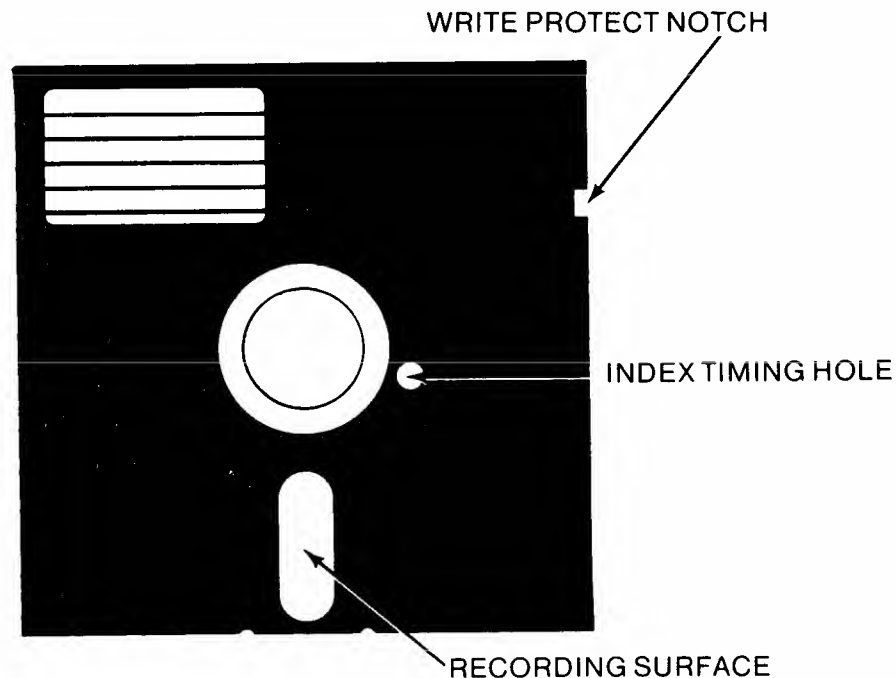
is similar to the record/playback head of a cassette tape recorder, but it is much more delicate and sensitive. The read/write head is positioned over various Tracks of data on the diskettes, as shown in Fig. 1.1. The number of tracks a disk drive can handle varies with the manufacturer, but the currently available track-handling capacities for 5 1/4-inch drives are 35, 40, 77 and 80 tracks, single- or double-sided. Most drives can handle double-density operation if used with a double-density controller.

Figure 1.1 Diskette Data Tracks



The third element of the drive is the electronics that can send signals back to the floppy disk controller and tell it that the diskette currently in place cannot be written to because of a special 'write-protect tab' that is placed on the 'write-protect slot'.

Figure 1.2 *Diskette Envelope*



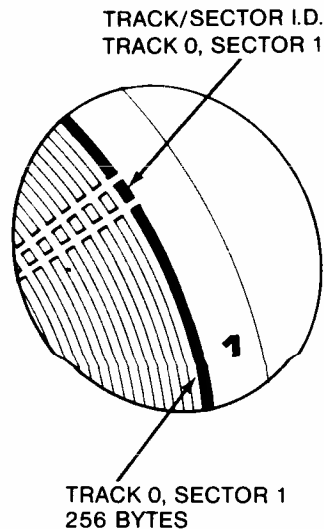
The drive can also tell the controller that the 'index hole detector' has detected the hole in the diskette known as the 'Index Hole'. This means the first byte of data on the track that the read/write head is currently placed over is just about to pass by the read/write head. The drive also has a 'select indicator light' (usually a red light-emitting diode) on the front to let you know when a drive is being accessed.

Tracks and Sectors

As you probably know, the term 'byte' means 8 particular 'bits' (or binary digits). The term 'sector' is the name given a particular group of bytes on a track.

A 'track' represents a group of sectors. All the sectors on a given track are generally referred to as 'the track', although other disk data that is used by the controller, but not normally accessed during sector I/O operations, is also stored on 'the track.'

Figure 1.3 *Diskette Tracks and Sectors*



Data is written and read a full sector at a time. After positioning the head over a desired track, the controller is commanded to read or write a sector. For every byte transferred, the controller turns on a bit in its Status Register that is called a 'data request' flag, or DRQ. When the DRQ goes on, the controller either desires the next byte to write during write operations or reads the next byte during read operations. The bytes are read or written to another address known as the 'data register.'

The standard sector length for the TRS-80 Model I is 256 bytes; however the FD1771 is capable of many other sector lengths.

A Technical Explanation of the FDC

The device housed inside the expansion interface which controls the disk drives (driven by your software) is the Western Digital FD1771B plastic or ceramic encased floppy disk controller IC chip.

In the TRS-80, the floppy disk controller is linked to the main processor (which is a Z-80) via the Z-80 memory address lines and data input/output lines. Since the floppy disk controller does not use the Z-80 port feature, it is considered to be Memory Mapped. This memory mapping technique is actually an advantage when you consider the TRS-80's slow (1.77 MHz) clock speed. This is because there are more Z-80 'opcodes' for memory addressing than there are for port addressing, such as BIT 0,(HL).

The channels that the Z-80 uses to communicate with the floppy disk controller 'registers' are addressed as memory locations just as the video and keyboard are. Addressing floppy disk controller registers is just like addressing any other memory location. For example:

```
LD A,(37ECH) ;READ THE CONTROLLER'S STATUS
```

The links between the Z-80 and the controller are made through eight 'data access lines' and the associated device control signals. The data access lines are used in transferring track and sector data, drive/diskette status, controller status and commands into and out of the FD1771 controller.

Addresses for Controlling the FDC

There are five dedicated memory locations for controlling the floppy disk controller. A memory WRITE means to write a byte to a memory address. A memory READ means to read a byte from the memory address. Below is an example of reading the controller's status.

```
LD A,(37ECH) ;READ CONTROLLER'S STATUS
```

The following will read the 'track' register:

```
LD A,(37EDH) ;READ TRACK REGISTER VALUE
```

Writing a byte address (**37EC**) issues a command byte to the Command Register. The following will write a **D0** to the controller, causing it to reset, and then read the controller's status.

Figure 1.4 *Controller Status Read Routine*

```
LD HL,37ECH ;HL-> COMMAND/STATUS REG
LD A,0D0H ;A= RESET COMMAND
LD (HL),A ;ISSUE VIA INDIRECT ADDRESSING
LD C,(HL) ;READ STATUS INTO REG C
```

The following addresses are linked to the various registers of the controller:

Figure 1.5 *Controller Register Addresses*

Address	OP-type	Bits Contain
37E0	WRITE TO	DESIRED DRIVE SELECT CODE
37EC	WRITE TO	DESIRED CONTROLLER COMMAND
37EC	READ FROM	CONTROLLER'S STATUS
37ED	WRITE/READ	CURRENT TRACK
37EE	WRITE/READ	LAST/DESIRED SECTOR
37EF	WRITE/READ	DATA BYTE TRANSFER, OR DESIRED TRACK

37E0 is the memory address that controls the selecting of the desired drive. Selection of a desired drive must be done before any operation can take place on it, whether that be positioning the read/write head, formatting a track, or reading and writing sectors. Reading this address will *not* give you the select code that you issued to it. Full details on selecting drives are given in the next chapter.

37EC is the memory address that links to the controller's Command/Status Register. When a Z-80 WRITE operation is made to this address, the byte written is loaded into the controller's Command Register, and the controller acts according to the command issued (if no operation is currently being performed). Otherwise, the command byte will be ignored unless it is a special FORCE INTERRUPT command which terminates the current operation.

Here is an example of issuing a controller command.

```
LD  A,0C0H    ;CONTROLLER CMD BYTE
LD  (37ECH),A ;ISSUE COMMAND
```

By executing a Z-80 read from the **37EC** memory address, the status of the current or last operation is returned from the controller's 8-bit Status Register. Each bit of the returned status indicates a status of something relating to the current function. Not all bits have the same meaning when reading the status of different disk functions.

Although the Command Register and the Status Register share this command memory address, they are independent controller registers. They share this common address for convenience. READS from the Command Register and WRITES to the Status Register are impossible. More on the Command and Status Registers will be presented later.

37ED is the memory address that links to the Track Register. This 8-bit Track Register holds the track number of the current position of the read/write head. Depending on some options, this register may be incremented (a fancy word for saying *add one*) each time the head is stepped in and is decremented (a fancy way of saying *subtract one*) when the head is stepped out. The Track Register may be written to and read from by accessing **37ED**.

An example of reading the Track Register is presented below.

```
LD  A,(37EDH) ;READ TRACK REGISTER
```

37EE is the memory address that links to the Sector Register. The 8-bit Sector Register holds the address of the desired sector to be read or written. The Sector Register may be written to and read from by accessing **37EE**.

Here is an example of writing to the Sector Register.

Figure 1.6 *Sector Register Write Routine*

```
LD      A,6          ;AMOUNT TO LOAD SECTOR
                        ;REGISTER WITH
LD      (37EEH),A    ;LOAD SECTOR REG
```

37EF is the memory address that links to the Data Register. This 8-bit Data Register is loaded with the byte to be written to disk (when called for by testing the *Data Request* bit in the Status Register) during a write operation. When doing a read operation, the Data Register contains the byte to be read by your software when the *Data Ready* bit is set in the Status Register.

Here is an example of reading the Data Register.

```
LD  A,(37EFH) ;READ DATA REGISTER
```

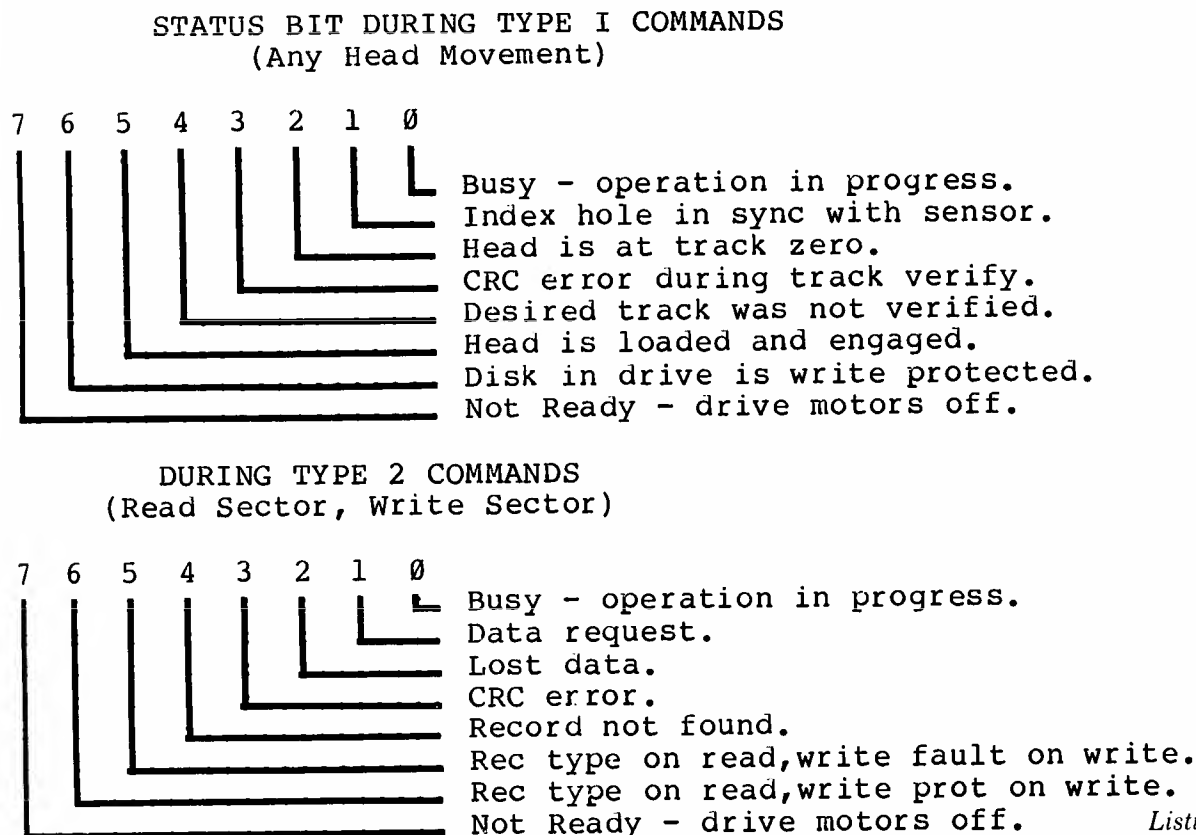
The Command/Status Register.

The Command and Status Registers are two independent FD1771B registers. As previously mentioned, reads are impossible from the Command Register, and writes are impossible to the Status Register. But of course you may read the Status Register and write to the Command Register.

The 8-bit Status Register holds the status information of the current disk operation. As previously mentioned, this register can be read but cannot be written to.

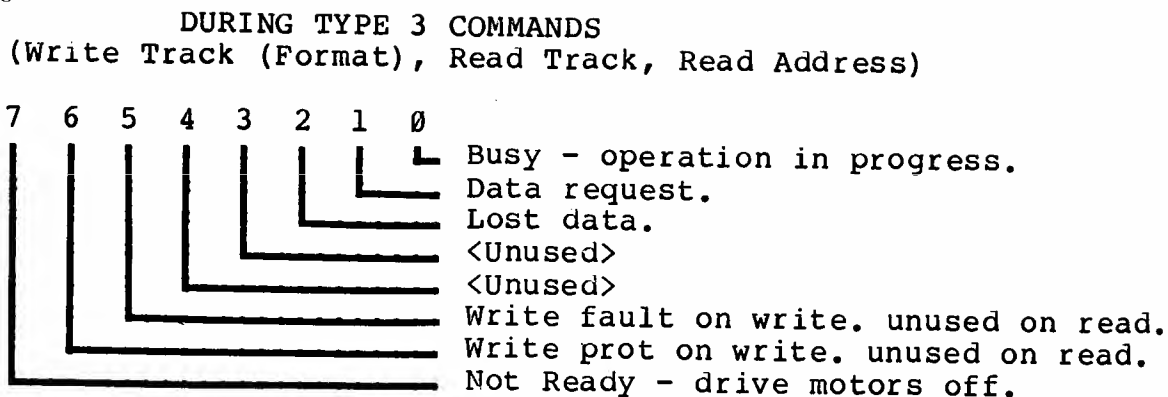
Here are the Status Register bits during and after the various types of operations:

Figure 1.7 Status Register Bits



Listing Continued...

...Continued Listing



TRS-80 Model I Disk Format

A particular operation known as WRITE TRACK, or FORMATTING, must be done to a track before sectors can be written and read by the floppy disk controller. Certain bytes are written to the diskette during this procedure and must be there when the diskette is accessed by the controller. The initialization of sectors and their *address identification bytes* is done during this process.

Figure 1.4 shows Western Digital's recommended track format. Some people run into problems with this disk format on a 5 1/4-inch floppy, because there just isn't enough room on the disk to put in all the overhead bytes plus the data. Most people don't know that the format recommended in the FD1771's data sheet is IBM format for 8-inch drives! All those overhead bytes are not needed, and can't fit on a 5 1/4-inch disk track.

These bytes are written once at the start of every track:

Figure 1.8 Western Digital Track Format

Number of Bytes	Hex Value of Byte Written
40	00 or FF
6	00
1	FC (index mark - track data starts here - unused by the floppy disk controller, but is useful when reading full tracks)
26	00 or FF

This group of bytes is written once per sector:

Figure 1.9 *Western Digital Sector Format*

6	00
1	FE (ID address mark - next 6 bytes are the sector address.)
1	Track number (0-FF)
1	Side number (0 or 1 - use 0 for TRS-80 stuff)
1	Sector number (0-EF, usually 0 to 9)
1	Sector length (should be 01 for 256 byte sectors)
1	F7 (this byte issued causes two CRC bytes of the 6 address byte to be written).
11	00 or FF
6	00
1	FB (data address mark - data comes now)
256	Sector data (IBM uses E5, so does TRSDOS)
1	F7 (cause two CRC's of the data to be written)
27	00 or FF

After the 10 sector-groups are written, write **FF**'s until the floppy disk controller goes to 'not busy status.'

The *Sector Length* byte is used by the floppy disk controller in computing how many bytes are in the sector when doing *sector* I/O. The floppy disk controller can perform two modes of computation of sector length. The first is called IBM format. If the *IBM Format Bit* is set in the sector read or write *Command Byte* issued to the floppy disk controller, the controller calculates the sector length to be as follows:

Figure 1.10 *Sector Length Calculation*

SECTOR LENGTH BYTE in Sector Address	Length of Sector in Bytes.
00	128
01	256
02	512
03	1024

If the IBM format bit is *not* set in the read/write command, here's how the floppy disk controller computes sector length:

$$\text{SECTOR LENGTH} = (\text{sector length byte} * 16)$$

If the sector length byte is zero, it is considered to be 256.

Here is a graphic description of how *non-IBM* formats are computed:

Figure 1.11 *Non-IBM Format Computation*

Sector Len Byte (Hex)	Number of Bytes in Sector (Decimal)
01	16
02	32
03	48
04	64
05	80
.	.
.	.
.	.
FE	4064
FF	4080
00	4096

This *non-IBM format* type calculation will give you sector lengths up to 4096 (256*16). The smaller the sector lengths, the more overhead it takes, and you lose data space. 256-byte sectors are pretty-much optimum for our use. TRSDOS uses standard IBM-compatible, 256-byte sectors, so the Sector Length byte is 1 when formatting, and the IBM format bit is set in the read/write sector command bytes when doing sector I/O.

The *Sector Data* bytes are the initial data that the sector contains. Have you ever looked at the sectors of a freshly formatted disk with some disk utility? That is what these bytes are. Any bytes except **F0 - FF** may be written as the initial sector data (TRSDOS uses **E5**). You may not use **F0 - FF** because when formatting (Write Track), the floppy disk controller uses these as control bytes. When doing *sector* writes, you may write anything you wish to the sector.

In Figure 1.4 did you notice all those overhead bytes? These bytes are used as 'padding.' Some of those bytes are **not** required, and do not have to be used. For example, you don't have to use those 40 bytes preceding the **FC** index mark, or the 26 bytes preceding the first sector, etc. Most of these bytes are unnecessary on 5 1/4-inch disks, and if written, you would not be able to fit 10 sectors on a track. Also, the Index Mark is not used by the controller.

Here are the formatting bytes I use for TRSDOS compatible sectors, and I've never had a problem using the following format.

These bytes are written once at the start of every track.

Figure 1.12 TRSDOS Track Format Bytes

Number of Bytes	Hex Value of Byte Written
1	FC (index mark-track data starts here-unused by the floppy disk controller, but is useful when dumping full tracks.
26	00 or FF

These Bytes are Written Once Per Sector.

Figure 1.13 TRSDOS Sector Format Bytes

Filler	
6	00
The Sector ADDRESS MARK	
1	FE (SECTOR ID address mark - next 6 bytes are the sector address.

Listing Continued . . .

...Continued Listing

The Sector ID

1	Track Number (0-FF)
1	Side Number (0 or 1 - Floppy disk controller ignores this byte)
1	Sector Number (0-FF)
1	SECTOR LENGTH (See text).
1	F7 (This byte issued causes two CRC bytes of the 6 address byte to be written.)

More Filler Padding

11	00 or FF
6	00

The Sector DATA ADDRESS Mark

1	FB (Data Address Mark - Data Comes Now)
---	---

User Data

(See Text) DATA BYTES (IBM Uses E5, So Does TRSDOS)

1	F7 (Cause Two CRC Bytes to be written after the data).
---	--

More Filler

27	00 Filler
----	-----------

After the 10 sectors are initialized, **FF**'s are to be written until the floppy disk controller goes to not-busy status.

The *only* required 'padding' bytes the controller needs in order to do sector access is 17 bytes between the sector ID and the sector data. The first 11 bytes are **FF**, and the last 6 bytes are **00**. Also, one byte of **00** must precede every sector ID address mark.

There is a disk formatting program for your use in Chapter 8.

Model III Supplement to Chapter 1

Model III Disk Controlling System

The Model III uses a Western Digital FD1793 for floppy disk controlling. The FD1793 floppy controller is virtually identical to the FD1771 except that it has double-density capability. Also, as matter of interest to hardware buffs, the FD1793 has a *true* bus instead of the *inverted* bus which the FD1771 uses. Other minor differences exist which will be explained as they come up.

The Model III does not use memory-mapped addresses for controlling the disk. Instead it uses Z-80 ports. The Model III has a number of other hardware features that the Model I does not have. These hardware features aid in controlling the disk, handling real-time clock interrupts and device interrupts from the RS-232 and 1500 baud cassette. Disk I/O is handled in a slightly different manner from the Model I.

The following is a list of the ports used in controlling the Model III disk.

Figure 1.14 *Model III Disk Control Ports*

Port (hex)	Data Flow	Use
F0	Read	Read the FDC status.
F0	Write	Issue an FDC command.
F1	Read/Write	FDC track register.
F2	Read/Write	FDC Sector register.
F3	Read/Write	FDC Data register.
F4	Write	Select drive and options.
E4	Write	Select non-maskable interrupt options.
E4	Read	Read non-maskable interrupt status.

Ports **F0** through **F3** correspond to the memory addresses **37EC** through **37EF** on the Model I. These are used to 'talk' directly to the controller.

The following is an example of reading the controller's status.

```
IN    A, (0F0H)      ;Read FDC status
LD    (0BF00H), A    ;Store in RAM
```

Port F4 is used in selecting the desired drive similar to the memory address 37E0 on the Model I. This port is also used in selecting other disk related features, as shown in the following list (all bits are assumed to be in the 1 state).

Figure 1.15 Port F4 Bit Summary

Port F4 Bit Summary.

Bit	Use
7	Select Double-density.
6	Generate Data Waits.
5	Set Write Precomp.
4	Select Side 1. (Not Used).
3	Select Drive 3. (Same as Model I)
2	Select Drive 2. (Same as Model I)
1	Select Drive 1. (Same as Model I)
0	Select Drive 0. (Same as Model I)

Or, you may address the port using register 'C'.

```
LD    C,0F0H      ;Load cmd/stat port number
IN     E,C         ;Read FDC status to E
LD     D,0D0H      ;Reset FDC command
OUT    (C),D       ;Issue reset command
```

Bit 7 is used in selecting the desired density of operation. Outputting a byte that has this bit set will cause double-density mode to be invoked, until a byte that has this bit reset is written, which causes single-density to be invoked.

Bit 6 is used to enable Z-80 waits during data I/O. This is used during sector and track I/O to prevent 'lost data' errors. This will be fully explained in a later chapter.

Bit 5 causes 200-microsecond write pre-compensation to occur during data writes. This should be set when writing to tracks 22 and above if you're in the double-density mode.

Bit 4 is potentially useful for selecting the second side of a drive. Radio Shack drives do not support dual sided operation.

Bits 3 through 0 are used in the same manner as the Model I to select the desired drive. Every time a drive is selected, the byte used for selecting must also contain the desired bit pattern to invoke the above options. This takes a little more programming than the Model I, but it is much more versatile.

The following is an example of selecting drive 0 with double-density set.

```
LD     A,81H      ;Drive 0 code, and DDEN
OUT    (0F4H),A   ;Select drive
```

Model III Non-maskable Interrupts

The Model III allows disk routines to utilize the Z-80 non-maskable interrupt (NMI) facility for disk I/O operations. This, coupled with the Z-80 Wait State function (described in the next chapter's supplement) makes it easy to use double-density without 'tricky' I/O loops. In my opinion, the Model III is an excellent machine when it comes to disk I/O.

The disk I/O non-maskable interrupt (NMI) feature allows two options to be independently selected. First, NMI occurs when an FDC interrupt request occurs, or in other words, when an FDC function is finished. And secondly, NMI occurs when the disk drive motors shut off or 'time-out'. These NMI features allow the elimination of testing for these conditions during disk I/O. This, combined with the Z-80 Wait State feature, allows smooth operation of double-density, without lost data errors due to slow processor speed. You might ask, "Why didn't Tandy simply increase the processor clock speed?" Tandy wanted to keep Model I compatibility with games and real-time software that was programmed to operate at the slow processor speed — although they are slightly different (1.77 vs. 2.0 MHz).

The non-maskable interrupt vector is located at **4049 - 404B**. You must have a jump to your interrupt routine at this vector when using the NMI feature.

Below are the ports used in selecting non-maskable interrupt disk functions.

Figure 1.16 *NMI Select Ports*

Port E4 - Write - Select Options.

Bit	Purpose
7	1=Cause NMI on FDC Interrupt (Function Done).
6	1=Cause NMI on Drive TIME-OUT.
5-0	Unused.

To select the desired options, you could load the 'A' register with the desired options set, and perform an OUT (0E4H),A.

Figure 1.17 NMI Status Read Routine

Port E4 - Read - Get NMI Status, Determine What Caused NMI.

7	0=FDC Interrupt Caused NMI.
6	0=Drives Timed-out.
5	0=Reset Button Pushed.
4-0	Unused.

Don't get excited about using bit 5 to detect the reset button. The ROM routine intercepts this before jumping to the NMI vector at 4049. The ROM routine jumps to its boot-strap loader if this condition occurs. Sorry about that!

Model III Double-density Disk Format

The Model III's single-density format is exactly the same as the Model I's single-density format. The formatting procedure is the same for single- or double-density, but the actual format is not. The sectors and overhead bytes are arranged the same, but the overhead bytes differ in value and quantity.

This format must be followed carefully for a correct format to be accomplished. Any deviation may lead to the last sector being chopped off, or sectors that can't be accessed — so follow it carefully. Before issuing a format command, you must set up the buffer *exactly* as the track is to be written.

Figure 1.18 Overhead Byte Format

Number of Bytes	Hex Value	Purpose
24	4E	Pre-track Filler
12	00	Pre-track Filler
3	F5	Pre-track Filler

This block is written *once* per sector. Since we are going to use the 18-sector scheme, this block will be written 18 times, with slight alterations to the Sector ID Block, i.e., sector name and track number. The blocks are written one right after the other.

Figure 1.19 *Sector ID Block*

1	FE	Sector ID Address Mark.
1	XX	Track Number. This MUST be Correct!
1	00	Side Number. Use 00.
1	XX	Sector Number. This Value Will be the Sector Name. E.I. 05 = SECTOR 5. See Text for Details
1	01	Sector Length Computation Value. See Text.
1	F7	Generate CRC Parity Bytes.
22	4E	Pre-sector Data Sector Filler.
12	00	Filler.
3	F5	Filler.
1	FB	Sector Data Mark. See Text.
256	E5	Initial Sector Data.
1	F7	Generate Sector Data CRC.
24	4E	Post Sector Filler.
12	00	Filler.
3	F5	Filler.

After all 18 sectors are written, approximately 500 bytes of 4E will have to be written, so format your buffer with about 600 bytes of FF just to be safe.

The 'Sector Name Byte' is the byte actually used in naming that sector. Model III TRSDOS uses sector names from 1 to 18. Other DOS's use 0 to 17. You could name the sectors in sequential order (such as 0, 1, 2, 3, etc.), but to speed-up sector accessing when doing sequential sector reads, the sector names should be staggered. I recommend staggering the sector names this way: 0, 6, 12, 1, 7, 13, 2, 8, 14, 3, 9, 15, 4, 10, 16, 5, 11, 17. This interlace technique allows more time for the DOS and/or programs to perform calculations before the next sector is accessed. In most cases, all 10 sectors can be accessed in 3 revolutions of the disk. It might otherwise take 20 revolutions because the programs were not quick enough to 'catch' the next sector as it was coming around.

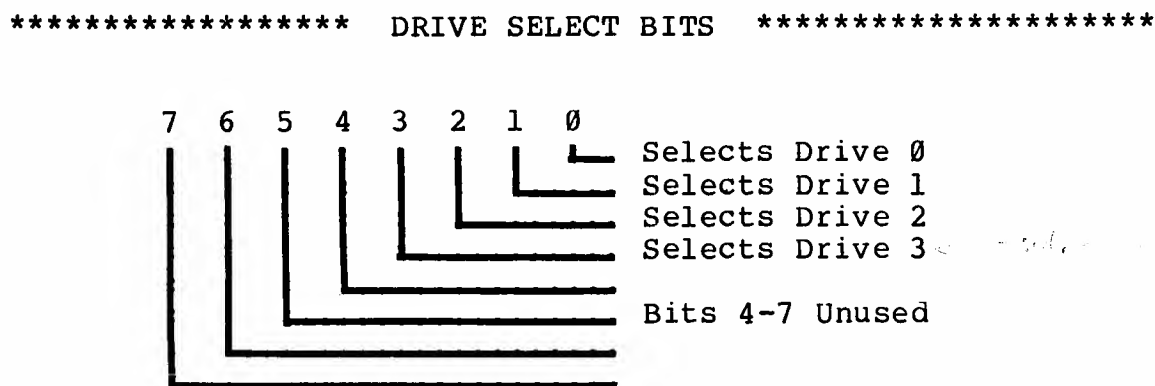
The 'Sector Length Computation' byte is used by the controller to determine how many bytes of data are in the sector. Using the IBM (normal TRSDOS) format, sector lengths of 128, 256, 512 and 1024 bytes can be achieved by using 0, 1, 2 and 3, respectively, for the Sector Length Computation value. There is no way to squeeze eighteen 512-byte sectors on a double-density track. In addition, if you decide that you want to use a different sector length, you must adjust the quantity of initial sector data bytes to the length of the sector. You will also have to do some experimenting to see how many sectors of such-and-such length you can fit on a track.

2

Selecting a Drive

In order to perform any disk I/O operations, the drives must first be 'selected.' Selecting a drive involves writing a byte (containing the proper *bit* turned on) to memory address **37E0**. This memory address is linked to the *disk drive select* circuitry. After the proper drive select code byte is written to **37E0**, the drive select circuitry inside the expansion interface fires up the disk drive motors and selects the proper drive. The drive that's selected is determined by the bits set in the *drive select code* byte that was written to **37E0**. Figure 2.1 displays the bit-to-drive relationship in selecting a drive. Since TRSDOS calls the drives: 0, 1, 2, and 3, and you are most likely used to calling them by those names, I will also refer to the drives that way.

Figure 2.1 *Drive Select Bits*



For example, if you wrote a **02** to **37E0**, this would fire all the drive motors, select the second drive in the system and reset the *not ready* bit in the Status Register. Software may test the 'not ready' bit to see if the drives are still rotating. In TRS-80 Model I system, the 'not ready' bit does not really mean that a drive is in or out of the system. In many systems the 'not ready' bit is used to test whether a selected drive is on the system. Don't ask me why the designers at Tandy didn't design the Model I to act this way. Anyway, when all the drives stop rotating, the 'not ready' bit will be set high again.

Every time a drive is selected, the motors of all the drives in the system are fired up, and the desired drive is selected. But keep in mind, this condition will only last for about 2.5 to 3 seconds. In order to keep the desired drive selected, and the motors running, when doing large amounts of data I/O or head-positioning functions that would last for more than 2.5 seconds, you must *re-select* the drive about every 2.5 seconds. Also, keep in mind that you may re-select a drive as often as you wish without any harmful effects to the function the controller is currently performing. You can verify that a drive has been selected by simply looking at the drive's *select-indicator-light* on the front of the selected drive's case. This is usually a red light-emitting diode (LED). The way your software can check to see if a drive is on the system will be discussed later in this chapter.

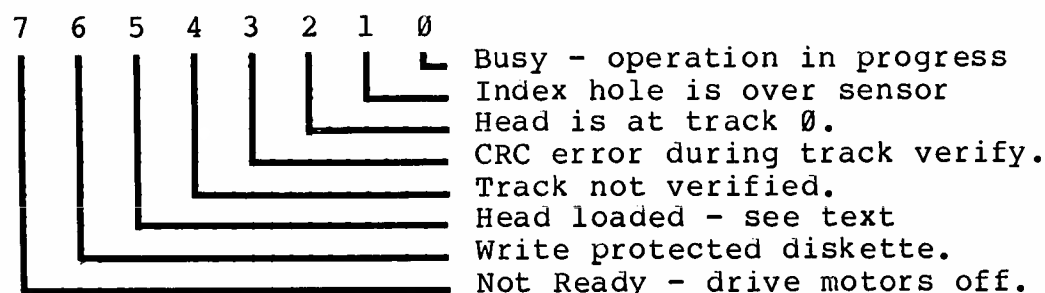
If a given drive is already selected, and you want to select another one, you don't have to wait until the drive motors stop. Just write a byte to **37E0** with the proper bit set (as shown in Fig. 2.1) for the drive you want to select. You will see the select-indicator light go out on the first one, and the new drive's select-indicator light will come on.

You *could* select more than one drive at a time by writing a byte to **37E0** that contains the select bits for all drives to be selected, but this is useless because none of the disk functions will work properly. This should *never* be done.

Reading the Selected Drive's Status

You may get the selected drive's status by performing a read from the memory address **37EC**, which is linked to the Status Register. Below are all the possible conditions of the Status bits during or after a Type I command (which is any head movement) or before any diskette I/O command is executed.

Figure 2.2 Status Bits During Type I Command



Bit 0 — The *busy* bit (0) means there is an operation in progress. For example, if you selected a desired drive and issued a RESTORE command (which restores the selected drive's head to track zero), you would not want to issue another command, such as the SEEK command (which positions the selected drive's head over a desired track) until the restore was completed. Your software should test the *busy* bit continuously until it goes to 0. Then you may issue the next command.

Bit 1 — When this bit is set, the index hole punched in the diskette is letting light pass through to the index sensor. This bit is useful in software for determining whether or not a selected drive is connected to the system.

Bit 2 — This bit is set when the read/write head is positioned over track zero.

Bit 3 — This bit is set if a CRC error was encountered when the controller read a sector ID in order to verify the track position.

Bit 4 — This bit is set if the track verification produced a *verify bad* condition. In other words, the head didn't end up on the desired track, or if it really *did* end up on the right *physical* track but the track byte in the sector ID was not correct, this too would cause a bad verify condition.

Bit 5 — This bit is set when the head is loaded. This means the head is pushed closer to the diskette than normal so the drive can read or write some data. In this system, this particular bit is ignored because the currently selected drive's read/write head is always loaded immediately.

Bit 6 — This bit is set if the diskette in the selected drive has a write-protect tab on it. When a diskette has a write-protect on it, it cannot be written to.

Bit 7 — This is the 'not ready' bit. This bit set means that all the drive motors are not rotating.

Hello, is Anybody Home?

Sometimes the need arises to determine whether or not a drive is hooked up to the system, or if a diskette is in the selected drive. This can be accomplished by testing the selected drive's index hole in a timed loop. If, in a given amount of time, the index bit never came on, we know that the drive is not there. If, within a given amount of time the index bit is always on, we know that the drive is there, but there is no disk in it. But if we test the index bit for a given amount of time and it *toggles*, we know that the drive is on the system, a diskette is placed inside of it, the drive door is shut and we may access the disk (assuming it is formatted properly).

In order to determine how long to test the index bit with our software, we must know how often the index hole passes the index hole detector.

Even I can figure out that if the diskette is rotating at the rate of 300 RPM

(revolutions per minute), we can determine how many times the index hole aligns with the index hole detector by dividing 300 RPM by 60 seconds. What did you get? My calculator tells me that the index hole comes around 5 times per second. Did you understand the equation? Good. Let's continue.

Below is an assembly listing of a routine that returns with the *carry flag* set if the currently selected drive is not on the system or a diskette is not in it. The *carry flag* is reset if the drive is connected to the system, and a disk is properly mounted in it.

Figure 2.3 Carry Flag Set Routine

```

*****
**          CHECK 'ON-LINE' STATUS OF A DRIVE          **
*****

LD      A,0D0H          ;Controller Reset Cmd
LD      (37E0H),A       ;Clear Existing Status.
LD      A,2             ;Select Code for Drive 1
LD      (37E0H),A       ;Select Drive
LD      BC,2000H        ;Number of Times to Test
                        ; Index Hole - Aprx. 1/4 Sec.

*****
**      This Series of Loops/Calls Checks for the Index      **
**      Flag Status to Change.                               **
*****

L1      CALL    TEST          ;Test Index Bit
JR      NZ,L1                ;Loop if Index Bit Set.
L2      CALL    TEST          ;Test Index Bit
JR      Z,L2                 ;Loop if Index Bit is Reset
L3      CALL    TEST          ;Test Index Bit
JR      NZ,L3                ;Loop if Index Bit Set
XOR     A                   ;Reset Carry Flag - Drive is
                        ; On-line, a Disk is in it
                        ; and the Door is Shut.
RET                                ;Return From Test

*****
**      Subroutines L1-L3 Call This TEST Routine. This      **
**      Routine Checks BC for Zero. If BC = 0 Then That Means **
**      That One of the L1-L3 Calls Kept Looping and Exhausted **
**      BC, and an Error Return is Generated. Otherwise the **
**      Index Bit is Tested and the Routine Returns to L1-L3. **
*****
Listing Continued...

```

... Continued Listing

TEST	DEC	BC	
	LD	A,C	;Dec Test Counter
	OR	B	;Get LSB of Counter
	JR	NZ,T1	;Test if BC=0
	SCF		;Go if Counter Not Zero
			;Set Error Flag
	POP	BC	; Drive is Not Ready
			;Kill Internal Call from
			; L1, L2, or L3
	RET		;Return to Original Caller
T1	LD	A,(37ECH)	;Get Controller Status
	BIT	1,A	;Test Index Bit
	RET		;Return to L1, L2 or L3

Model III Supplement to Chapter 2

Selecting Model III Drives

Port F4 is used in selecting the desired drive similar to the memory address 37E0 on the Model I. This port is also used in selecting other disk related features (see the following list). All bits are assumed in the 1 state.

Figure 2.4 *Port F4 Bit Summary*

Port F4 Bit Summary.

Bit	Use
7	Select Double-density.
6	Generate Data Waits.
5	Set Write Precomp.
4	Select Side 1. (Special Hardware)
3	Select Drive 3. (Same as Model I)
2	Select Drive 2. (Same as Model I)
1	Select Drive 1. (Same as Model I)
0	Select Drive 0. (Same as Model I)

Bit 7 is used in selecting the desired density of operation. Writing a byte to Port F4 that has bit 7 set will cause the double-density mode to be invoked until a byte that has this bit reset is written — which causes single-density to be invoked.

Bit 6 is used to enable Z-80 waits during data I/O. This can be used during sector I/O or track I/O (formatting) to prevent lost data errors. This eliminates time wasted checking for data requests and busy during data I/O, thus eliminating lost data errors.

When the 'wait bit' is set while selecting a drive, the Z-80 will go into a memory wait state when trying to fetch the next instruction from memory (the next M1 cycle). This condition will be terminated when one of the following conditions occurs:

1. If an FDC interrupt request is generated after a function is finished.
2. If an FDC data request occurs.
3. If the reset button is pressed.
4. If the Z-80 is in the wait state for more than 1024 microseconds (1.024 milliseconds). This last termination is provided so that no chance of memory loss occurs due to insufficient memory refresh. How all this applies to disk I/O will be explained in the Chapter 4 supplement.

Bit 5 causes 200-microsecond write pre-compensation to occur during data writes. This should be set when writing to track 22 and above when in double-density mode.

Bit 4 is potentially useful for selecting the second side of a drive; however, Tandy does not support dual-sided drive operation as of this writing.

Bits 3 through 0 are used in the same manner as the Model I. These are used to select the desired drive. Every time a drive is selected, the byte used for selecting must also contain the desired bits set to invoke the above options. This takes a little more programming than the Model I, but it is much more versatile.

The following is an example of selecting drive 0, with double-density set.

```
LD    A,81H          ;Drive 0 code, and DDEN
OUT   (0F4H),A       ;Select drive
```

Checking The On-Line Status

To check whether the drive is on-line with a disk inserted in it, you may use the following program.

Figure 2.5 On-Line Drive Status Routine

```
*****
**                                **
**          CHECK ON-LINE DRIVE STATUS          **
**                                **
*****

LD      A,0D0H          ;FDC Reset Command
OUT     (0F0H),A        ;Reset FDC. Puts FDC in Mode 1.
LD      A,1             ;Select Code for Drive 0
OUT     (0F4H),A        ;Select Drive 0
LD      BC,2400H        ;# of Times to Check
L1      CALL TEST        ;Get Drive Status
JR      NZ,L1           ;Loop if Index Hole Detected.
L2      CALL TEST        ;Get Drive Status
JR      Z,L2            ;Loop if Index Hole Not Detected
L3      CALL TEST        ;Get Drive Status
JR      NZ,L3           ;Loop if Index Hole is Detected.
XOR     A               ;No Error
RET     A               ;Return

*****
** Subroutines L1, L2 and L3 Call This Routine to Get the **
** Index Hole Detection Status. NZ Returned Means the **
** Index Hole is Being Detected. Otherwise Z is Set.      **
*****

TEST    DEC      BC      ;Decrement 'Status Test' counter.
LD      A,C          ;Get LSB of Counter.
OR      B            ;Is BC Zero?
JR      NZ,TEST1     ;No. Get Status and Return.
POP     BC           ;Clear Call from L1, L2 or L3.
SCF     ;Carry Set Means Drive Not Ready
RET     ;Return to caller
```

Listing Continued . . .

... Continued Listing

TEST1	IN	A, (0F0H)	;Get FDC Status.
	BIT	1,A	;Test Index Hole Sense.
	RET		;Return to L1, L2 or L3.

3

Type I Commands — Head positioning

This chapter deals with the drive's read/write head positioning commands that are issued to the controller's Command Register via 37EC. Head movement operations position the selected drive's read/write head over the various *tracks* of data.

These head positioning commands are referred to as Type I commands, and include the RESTORE, SEEK, STEP, STEP-IN, STEP-OUT and FORCE INTERRUPT commands. All of these Type I commands, except the FORCE INTERRUPT command (which resets the FDC), contain a parameter called the STEPPING RATE field, which is represented by r1 and r0 in the command format diagrams as bits 1 and 0. This field determines how many milliseconds the controller should delay after each head movement before resetting the Busy flag in the Status Register, or issuing another step pulse (depending on the operation). This delay allows the currently selected drive's read/write head to stabilize over each track before performing data transfers or issuing another head movement pulse. After performing an operation that steps the read/write head just once, the Busy flag in the Status Register does not get reset until the specified delay is finished.

Figure 3.1 depicts the 4 possible stepping delay values. These two bits are set accordingly in the Type I command byte that is issued to the Command Register via 37EC.

Figure 3.1 Stepping Delay Values

Bit 1	Bit 0	Stepping Delay Between STEPs in Milliseconds
0	0	6
0	1	12
1	0	20
1	1	40 (Normal TRSDOS rate)

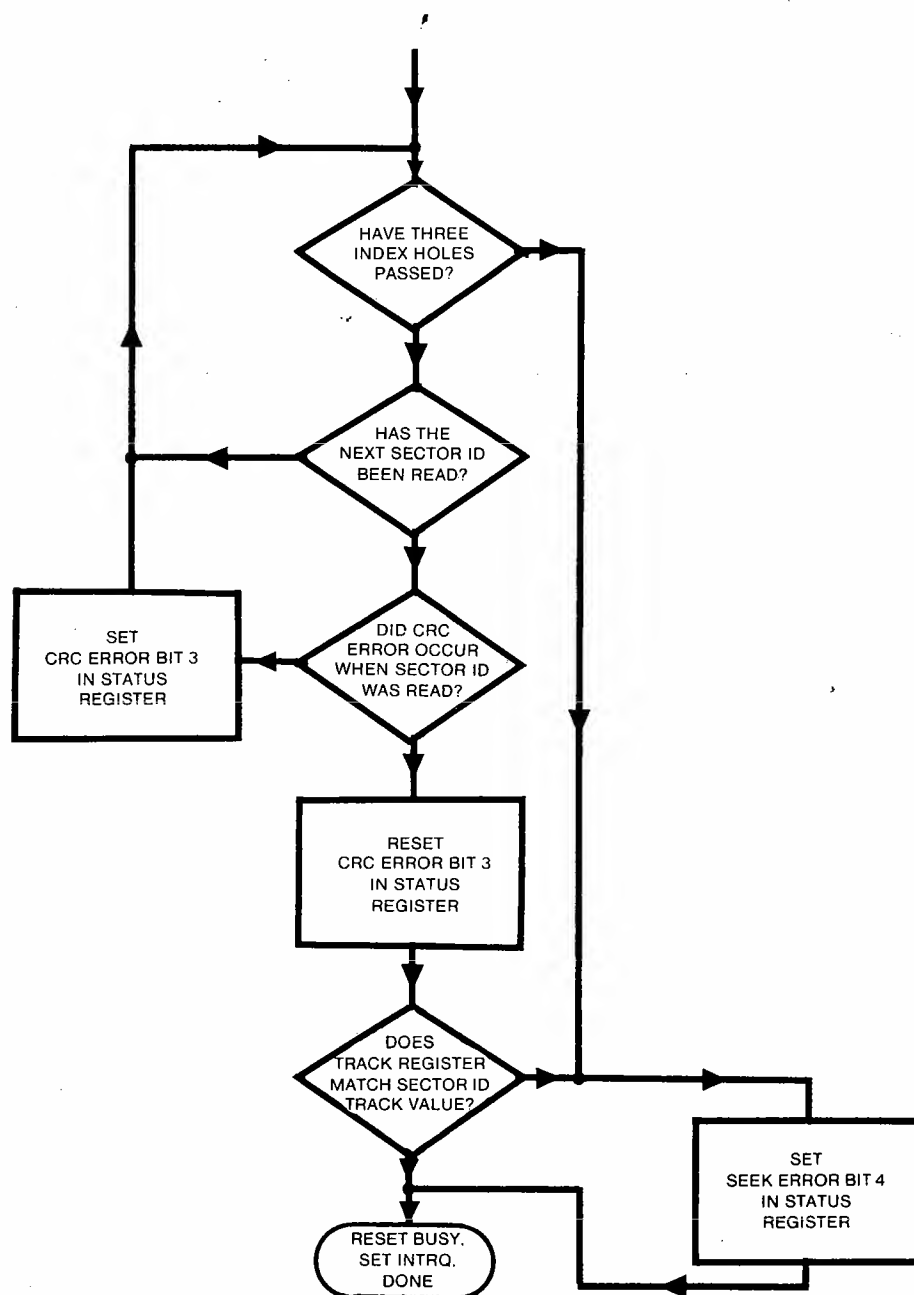
The read/write head-stepping mechanism in the older Radio Shack disk drives made by Shugart (SA-400), can only handle the slow 40-millisecond rate, but other drives, such as the MPI, Vista, Tandon, or the newer Radio Shack disk (Tandon design) drives, can handle the much faster 6-millisecond rate (see the 'stepping rates' chart in the back of the book).

All of the Type I commands use bit 3 as the *Head Load flag* in the command byte that's issued to the controller via 37EC. It will be represented as the 'H' field in each of the Type I command format diagrams. In other systems, the Head Load bit is set if the read/write head is to be LOADED (moved closer to the diskette in preparation to read or write data to the diskette) at the beginning of the command. The Head Load bit is reset in the command bytes if the head is not to be loaded at the beginning of this command.

The Head Load bit can usually be ignored in the normal Model I system because the selected drive's head is *always* loaded right after being selected, and on some drives (like the Tandons) the head is loaded when the drive door is shut. You might ask, "Why is this Head Load function in the controller in the first place?" One reason is that when doing head movement, it is not necessary to load the head until a data transfer is to take place. It takes approximately 10 milliseconds for the drive's read/write head to stabilize after loading it. Anyway, you can ignore this bit in this system because the read/write head is always loaded in the selected drive. The Head Load feature was built into the controller primarily for the use with 8-inch drives, because in 8-inch drive systems, the diskettes are constantly rotating. This could cause undue wear on the drive's read/write head if it were always loaded.

All the Type I commands, except the FORCE INTERRUPT command, use bit 2 of the command issued to the controller as the *Verify flag*. This is denoted as 'V' in the command format diagrams. If this bit is set in the command byte, the controller will verify this particular Type I command. In other words, it will verify that the head is over the track contained in the Track Register.

When a verify is to take place, the first encountered sector ID is read from the diskette. The track byte from the sector ID is compared to the value in the Track Register. If there is a match and no CRC error occurred when reading the sector ID, the verify is OK and the operation is completed with no error bits set in the Status Register. If the track in the sector ID that was read from the diskette does **not** match the value in the Track Register, and the CRC of the ID was valid, an interrupt is generated, the *Seek Error* bit is set, and the Busy bit is reset in the Status Register. If the track in the sector ID that was read for the diskette *does* match the value in the Track Register, but the CRC of the ID was bad, the 'CRC Error' bit in the Status Register is set, and the next encountered sector ID is read for verification. If an ID field with a valid CRC cannot be found within 2 diskette revolutions (400 milliseconds), the controller terminates the operation, generates an interrupt, and sets the CRC Error bit in the Status Register.

Figure 3.2 *Flow Diagram of a Verify*

The STEP-IN Command

When the controller receives the STEP-IN command, it steps the selected drive once in the direction of the highest numbered track. For example, if the head was positioned over track 17 and you issued the STEP-IN command, the controller would position the read/write head one track higher, making the read/write head's new position track 18.

After a delay determined by the stepping rate field is done, a verification takes place if the 'V' flag was set in the command byte. An interrupt is generated at the end of this command.

Below is the format for the STEP-IN command.

Figure 3.6 *STEP-IN Format*

Bit:	7	6	5	4	3	2	1	0
	0	1	0	U	H	V	r1	r0

Bits 7 - 5 tell the controller this is the STEP-IN command.

Bit 4 is the Update Track Register flag.

Bit 3 is the Head Load flag.

Bit 2 is the Verify flag.

Bits 1 & 0 are the Stepping Rate field.

For example, issuing the STEP-IN command with the format of 01010111 binary, or **53** would be interpreted by the controller as follows:

1. This is the STEP-IN command.
2. Do not load read/write head at the start of the RESTORE operation.
3. Do not verify the restore with a sector ID from the new track position.
4. Update the Track Register by adding one to the current value.
5. Step head in one track toward the highest number track and delay 40-microseconds before resetting Busy and terminating operation.

Figure 3.7 STEP-IN Flow Diagram

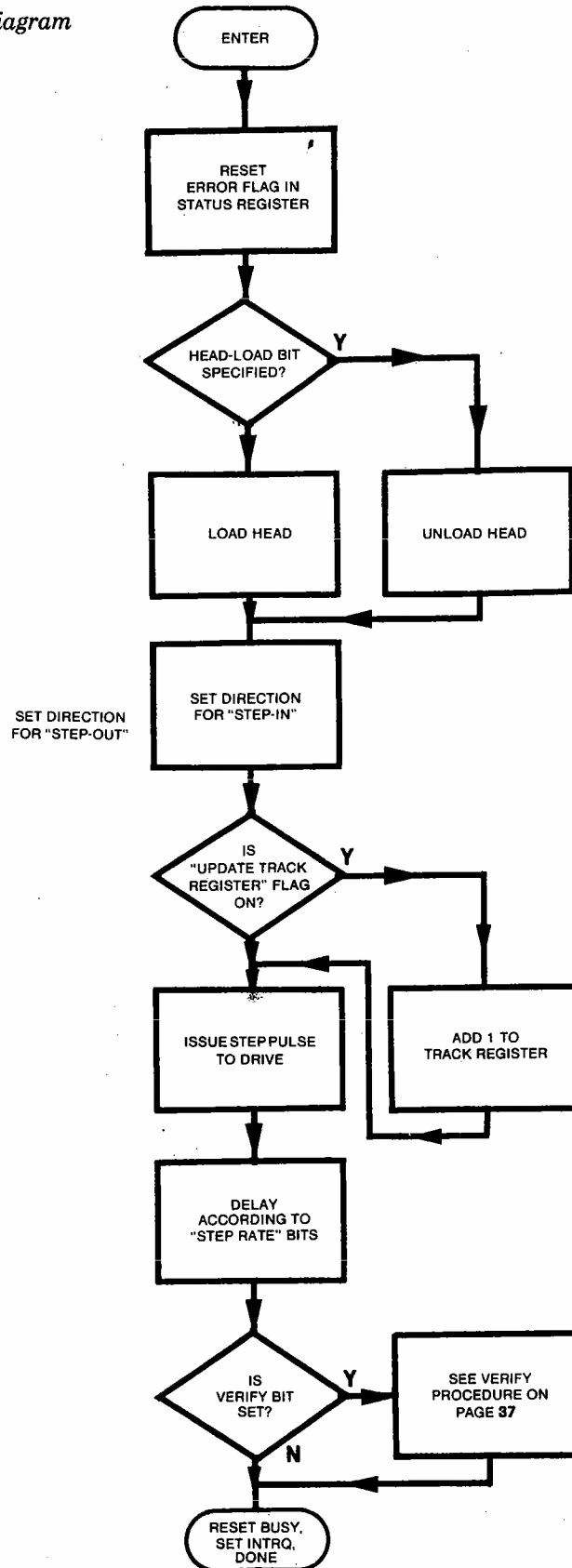


Figure 3.8 *Drive 0 STEP-IN Routine*

```

*****
**   Example of Stepping in Drive 0's Head 10 Times.  **
*****

START  LD      A,1                ;Drive Zero Select Code
        LD      (37E0H),A        ;Select Drive 0
        LD      B,10             ;# of Times to Step-in
        CALL    BUSY             ;Check if Controller Busy
L1      LD      A,53H             ;Step-in, Update Track Reg.
        LD      (37ECH),A        ; No Verify, 40 Ms Delay
        PUSH    HL               ;Issue Command
        POP     HL               ;Let Controller Set Up
        PUSH    HL
        POP     HL
        CALL    BUSY             ;Call Until Step-in is Done
        DJNZ    L1              ;Do Ten Times
        RET                     ;Return to Caller
BUSY    LD      A,(37ECH)         ;Get Controller Status
        RRCA                    ;Shift Busy into Carry Flag
        RET     NC               ;Ret If Not Busy
        JR      BUSY             ;Loop Till Not Busy

```

The STEP-OUT command

The STEP-OUT is almost identical to the STEP-IN command except that the read/write head of the selected drive is moved out one track toward track 0.

Below is the format for the STEP-OUT command.

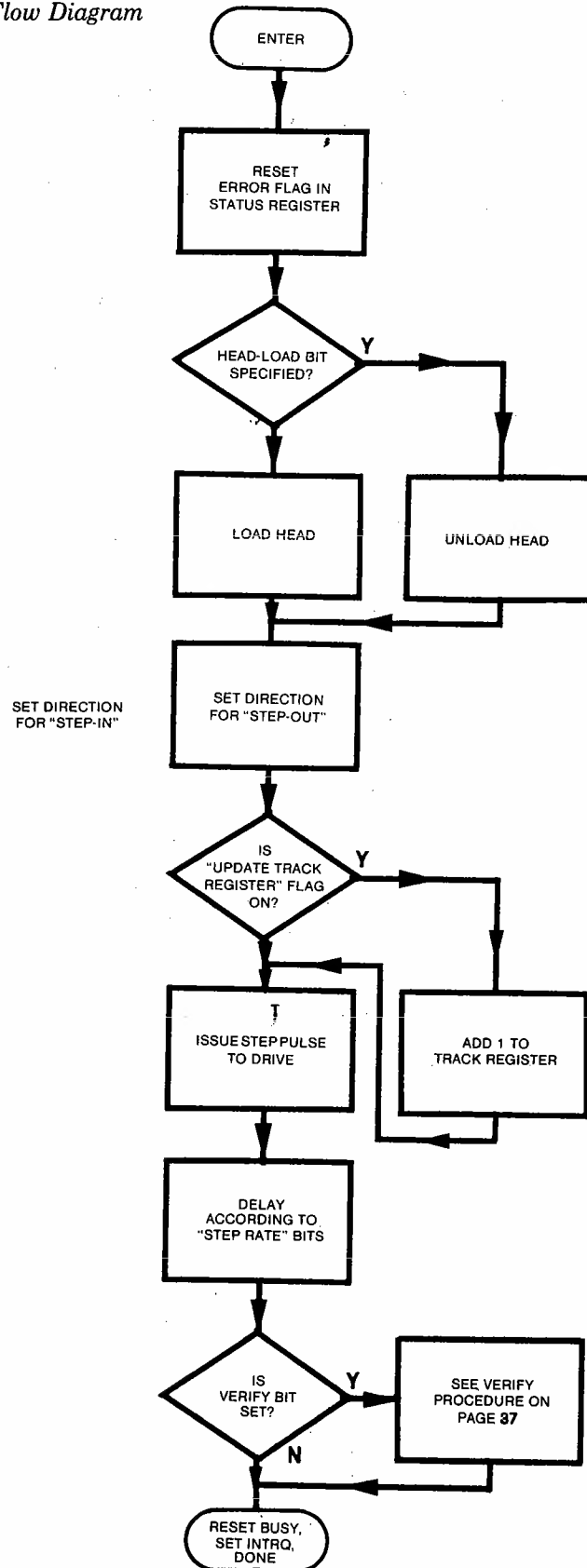
Figure 3.9 *STEP-OUT Format*

Bit	7	6	5	4	3	2	1	0
	0	1	1	U	H	V	r1	r0

For example, issuing the STEP-OUT command with the format of 01110000 binary, or 70 would be interpreted by the controller as follows:

1. This is the STEP-OUT command.
2. Do not load read/write head at the start of the RESTORE operation.

Figure 3.10 STEP-OUT Flow Diagram



3. Do not verify the RESTORE with a sector ID from the new track position.
4. Update the Track Register by adding one to the current value.
5. Step head out toward track 0 and delay 6-milliseconds before resetting the Busy flag and terminating operation.

Figure 3.11 Drive 1 STEP-OUT Routine

```

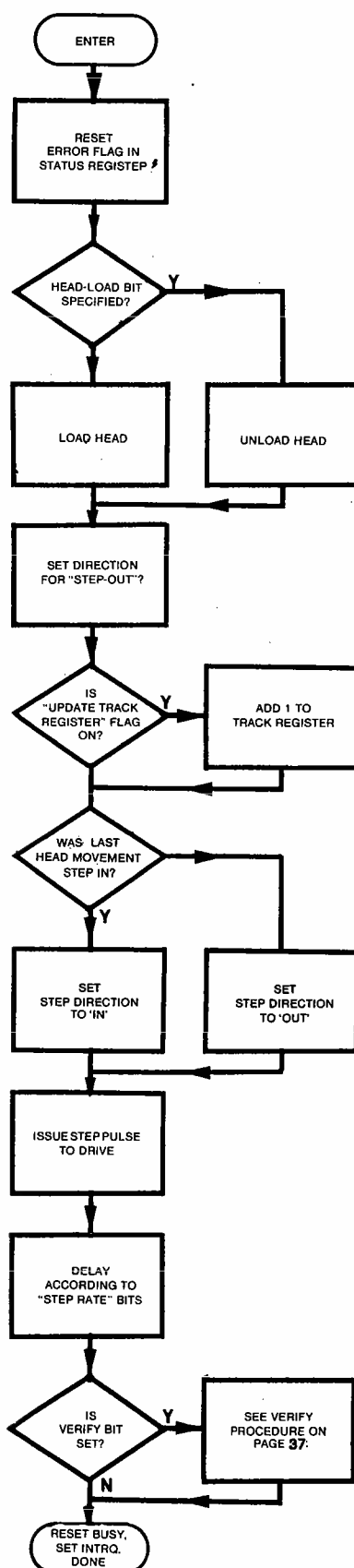
*****
** Demo of Stepping Drive One's Head Out 13 Times. **
*****

L1      LD      A,2           ;Drive 1 Select Code
        LD      (37E0H),A     ;Select Drive 1
        LD      B,13         ;# of Times to Step-out
        CALL    BUSY         ;Check if Controller Busy
        LD      A,73H        ;Step-out, Update Track Reg.
                                ; No Verify, 40 Ms Delay
        LD      (37ECH),A     ;Issue Command
        PUSH    HL           ;Let Controller Set Up
        POP     HL
        PUSH    HL
        POP     HL
        CALL    BUSY         ;Call Until Step-in is Done
        DJNZ    L1           ;Do Ten Times
        RET             ;Return to Caller
BUSY    LD      A,(37ECH)     ;Get Controller Status
        RRCA              ;Shift Busy into Carry Flag
        RET     NC          ;Ret if Not Busy
        JR      BUSY        ;Loop Till Not Busy

```

The STEP Command

The STEP operation is similar to the STEP-IN or STEP-OUT command, but it causes the controller to STEP the selected drive's read/write head one track in the *last* direction stepped, whether it was in or out. For example, if the head is positioned over track 19 and you issue a STEP-IN command, the controller will position the head over track 20. Now if you issue a STEP command, the controller will move the head one step in the last direction stepped, so it will now be positioned over track 21.

Figure 3.12 STEP Flow Diagram

Below is the format for the STEP command.

Figure 3.13 *STEP Format*

Bit	7	6	5	4	3	2	1	0
	0	0	1	U	H	V	r1	r0

For example, issuing the STEP command with the format of 00100000 binary, or **20** would be interpreted by the controller as follows:

1. This is the STEP command.
 2. Do not load read/write head at the start of the RESTORE operation.
 3. Do not verify the STEP with a sector ID from the new track position.
 4. Update the Track Register by adding or subtracting the appropriate value to the current value in the Track Register.
 5. Delay 6-milliseconds before resetting the Busy flag and terminating operation.
-

Figure 3.14 *Drive 2 STEP Routine*

```

*****
**          Here is an Example of Performing a STEP          *
**          Operation 3 Times on Drive 2.                    *
*****

START  LD      A,4          ;Drive 2 Select Code
        LD      (37E0H),A   ;Select Drive 2
        LD      B,3        ;# of Times to Step
        CALL    BUSY        ;Check if Controller Busy
L1     LD      A,20H        ;STEP, No Track Reg Update
        ; No Verify, 6 Ms Delay
        LD      (37ECH),A   ;Issue Command
        PUSH    HL          ;Let Controller Set Up
        POP     HL
        PUSH    HL
        POP     HL
        CALL    BUSY        ;Call Until Step is Done
        DJNZ    L1          ;Do Ten Times
        RET          ;Return to Caller
BUSY   LD      A,(37ECH)    ;Get Controller Status
        RRCA          ;Shift Busy into Carry Flag
        RET     NC        ;Ret if Not Busy
        JR      BUSY      ;Loop Till Not Busy

```

The SEEK Command

This is the most powerful head positioning command the controller has. What this command does is position the read/write head of the currently selected drive to the track contained in the Data Register. Before issuing the SEEK command, the Track Register must contain the selected drive's current track position. Then write to the Data Register (via 37EF) the track you want to position the head over. Now issue the SEEK command. The controller will step the read/write head in the proper direction until the contents of the Track Register are equal to the desired track you wrote to the Data Register. Updating is automatically done to the Track Register, and you have no control over this. Verification is done if the 'V' flag is set. An interrupt is generated at the completion of this command.

Below is the format of the SEEK command.

Figure 3.16 *SEEK Format*

Bits	7	6	5	4	3	2	1	0
	0	0	0	1	H	V	rl	r0

Bits 7-4 tell the controller this is the SEEK command.
 Bit 3 is the Head Load flag (unused).
 Bit 2 is the Verify flag.
 Bits 1 and 0 are the stepping rate field.

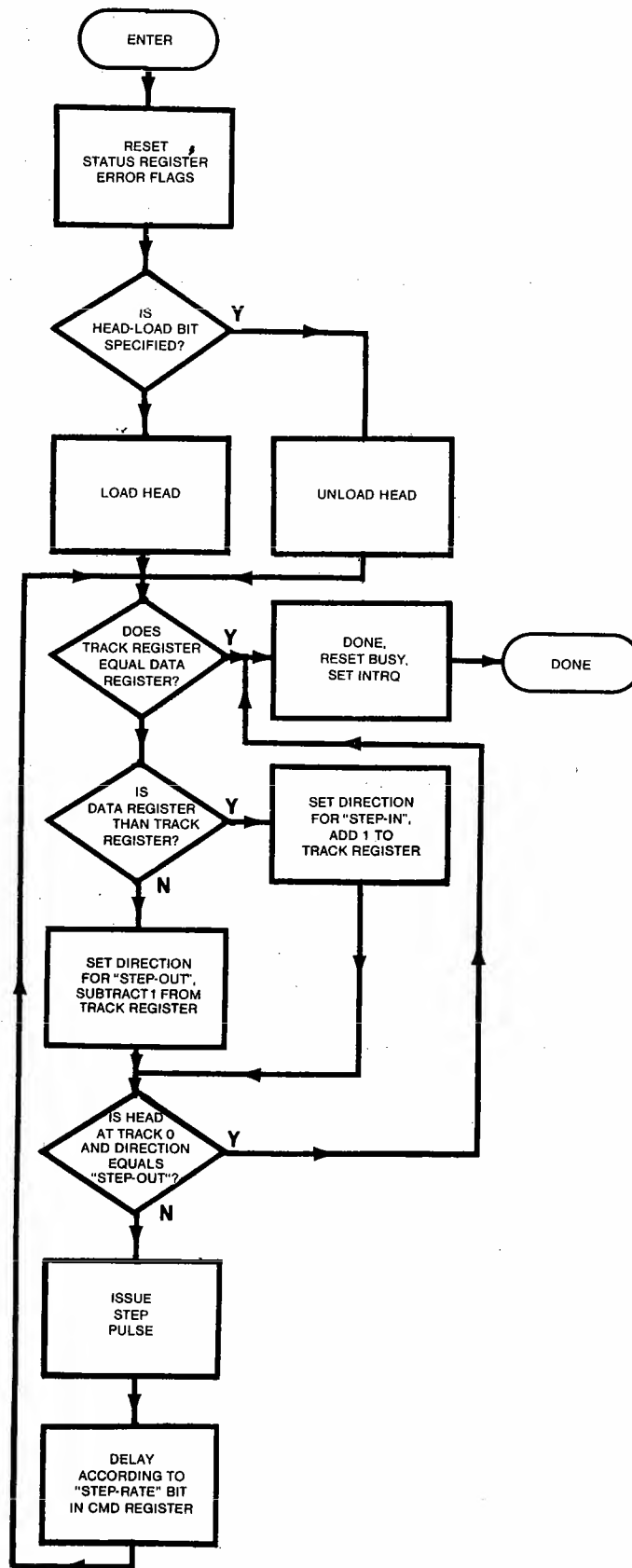
For example, issuing the SEEK command with the format of 00010111 binary, or 13 would be interpreted by the controller as follows:

1. This is the SEEK command.
 2. Do not load read/write head at the start of the RESTORE operation.
 3. Verify the STEP with a sector ID from the new track position.
 4. Delay 40-milliseconds between each stepping pulse before resetting the busy flag or terminating operation when done SEEKing.
-

Figure 3.17 *Drive 0 SEEK Routine*

```
*****
**      Example of Using the SEEK Command on Drive 0. A      **
**      Restore is Executed First So We Know What Track      **
**      the Head is Over Before Issuing the SEEK Command.    **
*****
```

Listing Continued . . .

Figure 3.15 *SEEK Flow Diagram*

... Continued Listing

```

START  LD      A,1           ;Drive 0 Select Code
        LD      (37E0H),A    ;Select Drive 0
        CALL    BUSY         ;Make Sure Controller is
                                ; Not Busy
        LD      A,3         ;Restore at 40 Ms Rate
        LD      (37ECH),A    ;Issue Command
        PUSH    HL          ;Let Controller Respond to
        POP     HL          ; Restore Command
        PUSH    HL
        POP     HL
f       CALL    BUSY         ;Wait Till Done
        XOR     A           ;Clr A
*---->>> LD      (37EDH),A    ;Ld Track Reg With Current Tk
        LD      A,13H       ;SEEK Command
        LD      (37ECH),A    ;Issue Command
        PUSH    HL          ;Wait for controller
        POP     HL
        PUSH    HL
        POP     HL
        CALL    BUSY         ;Wait Till Done
f       RET      ;Return to Caller
BUSY    LD      A,(37ECH)     ;Get Controller Stat
        RRCA      ;Put Bit 0 into C Flag
        RET      NC         ;Ret if Not Busy
        JR      BUSY        ;Loop Till Not Busy

```

The RESTORE function will automatically update the Track Register to 0, but I did it too, to illustrate how the Track Register must be loaded with the read/write head's current track position before issuing the SEEK command.

The FORCE INTERRUPT Command

This command is used to terminate the current operation being executed by the controller. It is good practice to do this before and after any disk READ or WRITE is performed, in order to reset the controller.

After issuing the FORCE INTERRUPT command to the controller via the Command Register at 37EC, the current function will be terminated, and Busy will be reset.

Figure 3.18 Forced Termination Routine

```

*****
**                               Example of a Forced Termination.                               **
*****

START  LD      A,0D0H        ;Force Interrupt Command
        LD      (37ECH),A    ;Issue Force Int Command
        RET      ;Ret

```

Model III Supplement to Chapter 3

Model III Head Positioning

Head positioning on the Model III is handled the same way as the Model I, except the addresses used are ports and not memory locations. Also, Unless you want to branch to a NMI routine after an FDC function is done, you must turn off the NMI options.

Below is an example of a RESTORE operation on a Model III.

Figure 3.19 *Model III Restore Routine*

```

*****
**                Restore a Drive's Head to Track 0                **
*****

START    CALL     BUSY            ;Make Sure Controller is Not Busy
          XOR      A              ;A=0
          OUT      (0E4H),A        ;Turn Off NMI Options
          LD       A,1            ;Drive 0 Select Code
          OUT      (0F4H),A        ;Select Drive 0
          LD       A,0            ;Restore Cmd @ 6 Ms Step Rate
          OUT      (0F0H),A        ;Issue Restore
          CALL     DELAY           ;Wait for Controller to React
                                   ; Before Testing
BUSY      IN       A,(0F0H)        ;Get FDC Status.
          RRCA                     ;Shift Busy into Carry.
          RET      NC             ;Return if Not Busy
          JR       BUSY          ; Else Loop.
DELAY     EX       (SP),HL        ;Waste Time
          EX       (SP),HL
          EX       (SP),HL
          EX       (SP),HL
          RET                      ;Return

```

4

Disk Data Input/Output Commands

This chapter deals with the controller commands used in reading and writing data to the diskette. Data I/O techniques will be discussed in Chapter 5, where I guide you through the sector I/O driver called 'DISKIO/ASM.'

Programmed I/O

In some computer systems, including the Model II, data input and output (called I/O) is done by DIRECT MEMORY ACCESS (DMA). This means that the software tells the controller what operation is to be done, how many bytes are to be transferred, and where in memory the data is to be written to or read from (called a 'buffer'). But the Direct Memory Access Controller actually performs the operation and reads/writes directly to/from memory the bytes involved without the Z-80's help. No software is required to get or put a byte each time one is needed.

The two main advantages of Direct Memory Access are the extremely fast transfer potential and the simplicity of the software required to run it.

The TRS-80 Model I does its disk I/O by a method called 'Programmed I/O.' The name is such because the 'program' (or software) handles the transfer of each individual byte into, or out of the data I/O buffer.

When an operation is to be done, the software tells the controller what operation is to take place. When a byte is to be read or written, the controller sets a 'DATA REQUEST (DRQ)' flag or bit in its Status Register. The software must test this flag at regular intervals, in some form of loop, in order to know when a byte is to be read or written. The Busy bit must also be tested to determine when the operation is completed.

In read operations, when the DRQ flag is set, the Data Register contains the byte just read from the disk. The software reads this byte from the Data Register.

In a write operation, the software supplies the Data Register with the next byte to be written to the disk.

If the software fails to keep up with the Data Request Bit, the *Lost Data* bit is set in the Status Register, and the operation is terminated. 2.

In order to determine whether a command may be given to the controller, or to tell whether a current command is finished, you must check the Busy flag in the Status Register.

Figure 4.1 *Busy Test Routine*

```
*****
** Here is an Assembly Source of a Typical Busy Test Loop **
*****

LOOP    LD      A,(37ECH)      ;Get Status from Status Reg.
        RRCA                    ;Shift Busy into Carry
        RET     NC             ;Return if Not Busy
        JR      LOOP          ;Try Until Not Busy
```

It could also be done this way.

```
START   LD      HL,37ECH      ;Cmd/Stat Register Location
LOOP    BIT     1,(HL)        ;Test Busy Bit
        JR      NZ,LOOP      ;Loop if Busy
        RET                      ;Ret
```

Or you might want to read the status into 'A' to test other status conditions.

Figure 4.2 *Status Test in A Register*

```
START   LD      HL,37ECH      ;Cmd/Stat Register
LOOP    LD      A,(HL)        ;Get Status
        BIT     1,A           ;Test Busy
        JR      NZ,LOOP      ;Go if Busy
        BIT     7,A           ;Check for Not Ready - Drive
                                ;Motors Off
        RET     NZ           ;Ret if Motors Off
        JR      LOOP          ;Get Status Again
```


Yes, I know I should have just tested the Not Ready bit without testing the Busy bit, but I just wanted to point out that you may address the controller's register addresses in any legal Z-80 way.

What the last routine did was:

Figure 4.3 *Status Test Loop*

```

START  1. Load HL With the Controller's Status Register.
LOOP   2. Read the Status into the A Register.
        3. Jump to LOOP if Controller was Busy.
        4. Jump to LOOP if Drive Motors are Still On.
        5. Return.

```

As described in Chapter 2, you must convert a logical drive number into its select code. In other words, the select codes for drives 0, 1, 2 and 3 are 1, 2, 4 and 8 respectively.

Here is a routine that will take the logical drive number in the B Register (0 to 3) and convert it to its Drive Select Code that will be written to the drive select latch at 37E0.

Figure 4.4 *Drive Select Code Routine*

```

*****
**      Routine to Convert a Drive to its Select Code.      **
*****

START  EQU      $           ;B = Drive 0,1,2 or 3
        INC     B           ;Make B = 1,2,3 or 4
        LD      A,80H       ;This Byte Gets Shifted
LOOP    RLCA              ;Shift A Left 1 (See Z-80 Man)
        DJNZ    LOOP        ;Do Until B = 0
        RET                     ;Return with Converted Value in
                                ; the A Register.

```

Do you remember (in Chapter 2) when we discussed how each bit in the drive select latch is related to each drive? If you are unsure, re-read that section.

The Read Address Command

When a diskette is formatted, each sector on any given track has a 6-byte header called the *Sector Address* or *Sector ID*. This is used by the controller to determine over what sector the head of the selected drive is about to be positioned. These sector address bytes contain the track number, sector name, byte length of sector, and a CRC of all these. You may read these bytes by issuing a READ ADDRESS command to the controller.

When the controller receives a READ ADDRESS command the Busy bit in the Status Register is set, and the *next* encountered Sector Address is read from the disk. The Data Request bit is set in the Status Register for each byte to be read (a total of six).

This is the order of the data bytes that are read by your software:

Figure 4.5 *Status Register Data Bytes*

Byte	Purpose
1	Track number - This is the track the sector is on.
2	Side number (usually 1 in TRSDOS) - can be ignored.
3	Sector name 00-FF.
4	Sector length - used by the controller in determining how many bytes long the sector is.
5-6	Two CRC bytes - Used in checking parity.

You may wish to read the next encountered address to determine what track the head is over on a particular drive.

The READ ADDRESS command is C0.

An example of a READ ADDRESS operation for drive 0 is given below.

Figure 4.6 *Drive Select Routine*

```
*****
**          Select Drive & Init Data Buffer That          **
**          Address is to Be placed in                    **
*****

START    LD      A,1                      ;For Drive 0
          LD      (37E0H),A                ;Select
          LD      BC,BUFFER                ;6 Byte Buffer to Store Bytes
          ; That are Read from Disk
          CALL    BUSY                     ;Make Sure Controller is Not
          ; Busy
```

Listing Continued . . .

... Continued Listing

```

*****
**          Issue Read Address Command & Wait          **
**          for Controller to Respond                  **
*****

        LD      A,0C0H          ;Read Address Command
        LD      (37ECH),A      ;Issue Command to Controller
        PUSH    HL              ;Wait for Controller
        POP     HL              ;
        PUSH    HL              ;
        POP     HL              ;

*****
**          Data Request Loop. Check for Byte to Get    **
**          and if Operation is Done                    **
*****

LOOP    LD      A,(37ECH)      13      ;Get Stat
        BIT     1,A            12      ;Check Data Request
        JR      NZ,GET         17 012  ;Go if Byte Ready
        RRCA                     4      ;Put Busy in Carry Flag
        RET     NC              15 → 11 ;Ret if Not Busy, Read Done
        JR      LOOP           12      ;Loop Back

*****
**          Address Data Byte Transfer                  **
*****

GET     LD      A,(37EFH)      13      ;Get Byte from Data Reg
        LD      (BC),A         7      ;Store in Buffer
        INC     BC              6      ;Bump Buffer Pointer
        JR      LOOP           12      ;Go Test Again

*****
**          Busy Test Loop                             **
*****

BUSY    LD      A,(37ECH)      13      ;Get Stat
        RRCA                     4      ;Shift Busy into Carry
        RET     NC              15 → 11 ;Ret if Not Busy
        JR      BUSY           12      ;Loop Till Not Busy

```

The READ TRACK Command

This command allows you to read an entire track of data, not just the user data, but *every* byte on it. It will dump all the overhead bytes used by the controller, plus the user data we call Sectors. Here is a situation in which this function could be useful: You just got VTOS 3.0, and you soon discover that sector 4 on track 0 just

can't be read. Well, I've heard some people say that this sector was left unformatted. "*Nonsense*," I exclaim! When the diskette was formatted, Mr. Cook simply named this sector something other than a normal 0-9 sector name (when formatting you may name your sectors anything you want from 00-FF). I think it was 7C or something like that. Regardless, all you have to do is read the entire track and look to see what that rascal was named. Also, I think M.S. Adventure does this to *all* of the sectors of their adventures so you can't copy the diskette with your operating system, nor can you 'superzappers' go looking through it. As a matter of fact, the *track* names on M.S. Adventures are different from normal too. That screws up the read/write head seeking if you verify your seeks. It's simple — just read the track, and you'll be able to see every single byte on it.

The READ TRACK command is E4.

Below is an example of a READ TRACK operation.

Figure 4.7 READ TRACK Routine

```

*****
**      Select Drive, Init Buffer to Put Track Data.      **
**      Test for Busy and Issue Command                  **
*****

START   LD      A,1                      ;For Drive 0
        LD      (37E0H),A                ;Select It
        LD      BC,BUFFER                ;Start of Buffer to Put Track
                                           ; Bytes That are Read
        CALL    BUSY                      17 ;Make Sure Controller Not Busy
        LD      A,0E4H                    ;Read Address Command
        LD      (37ECH),A                ;Issue Command
        PUSH    HL                        ;Wait for Controller
        POP     HL
        PUSH    HL
        POP     HL
LOOP    LD      A,(37ECH)                  13 ;Get Stat
        BIT     1,A                        12 ;Check Data Request
        JR      NZ,GET                     17 17 ;Go if Byte Ready
        RRCA                                4 ;Put Busy in Carry Flag
        RET     NC                         15 11 ;Ret if Not Busy, Read Done
        JR      LOOP                       12 ;Loop Back
GET     LD      A,(37EFH)                  13 ;Get Byte from Data Reg
        LD      (BC),A                     7 ;Store in Buffer
        INC     BC                         6 ;Bump Buffer Pointer
        JR      LOOP                       12 ;Go Test Again
BUSY    LD      A,(37ECH)                  13 ;Get Stat
        RRCA                                4 ;Shift Busy into Carry
        RET     NC                         15 11 ;Ret if Not Busy
        JR      BUSY                       12 ;Loop Till Not Busy

```

The WRITE TRACK or FORMAT Command

Formatting is just the opposite of reading a track. This is the command used in preparing a diskette for use. In this operation the creation and naming of sectors and tracks takes place.

The controller starts writing the format data as soon as the next Index Pulse is encountered. It keeps on writing until the Index Pulse is encountered again. See Chapter 1 for how your data must be set up for a track write.

It's possible you may never want to use the format operation, because under most conditions the TRSDOS, VTOS and NEWDOS formatters will work just fine, unless you want to name the sectors or tracks something non-standard or use a different sector length to link 128-byte sectors. There is a formatter program in the back of the book you may study and customize.

The WRITE TRACK command is: F4

Here is an example of writing a track:

Figure 4.8

WRITE TRACK Routine

```

*****
**      Select Drive and Init Data Buffer Pointer Start      **
*****

        LD      A,1                ;For Drive 0
        LD      (37E0H),A          ;Select
        LD      HL,37ECH           ;Cmd/Stat Register
        LD      BC,BUFFER          ;Start of Buffer That Contains
                                   ; All the Necessary Bytes
        CALL    BUSY               ;Make Sure Controller is Not
                                   ; Busy

*****
**      Issue Cmd and Get First Byte to Write              **
*****

        LD      A,0F4H             ;Write Track/Format Cmd
        LD      (37ECH),A          ;Issue Command
                                   ;The PUSH/POP Wait is
                                   ; Unnecessary When Formatting
        LD      A,(BC)             ;Get Next Byte to Write
        INC     BC                 ;Bump Buffer Pointer

*****
**      Data Request Test Loop                             **
*****

LOOP    BIT     1,(HL)             ;Check Data Request
        JR      NZ,PUT             ;Go if Byte Ready
        BIT     0,(HL)             ;Check if Write is Done
        RET     NC                 ;Ret if Not Busy, Read Done
        JR      LOOP              ;Loop Back

```

Listing Continued . . .

READ SECTOR Command

... Continued Listing

```
*****
**                Transfer Data Byte to Data Register                **
*****

PUT      LD      (37EFH),A      ;Write Byte
          INC     BC            ;Bump Buffer Pointer
          LD      A,(BC) ,      ;Get Next Byte to Write
          JR      LOOP         ;Go Test Again

*****
**                Busy Test Loop                                    **
*****

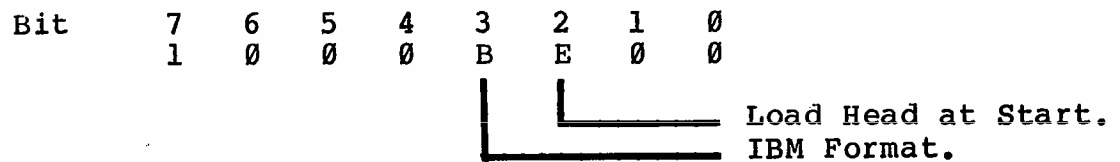
BUSY     BIT      0,(HL)        ;Busy?
          RET      Z            ;Ret if Not Busy
          JR      BUSY         ;Loop Till Not Busy
```

Read Sector command

This is the command issued before reading a sector. If the sector is not found within two revolutions of the diskette, the *Sector Not Found* bit is set, and the operation is terminated.

This is the format for the READ SECTOR command.

Figure 4.9 READ SECTOR Format



As in all TRS-80 Model I disk operations, the 'Load Head At Start' bit can be ignored because the head is loaded when a drive is selected.

The 'IBM Format' bit will be discussed in the section on formatting. This bit is normally set during this operation.

The usual byte issued is 88.

Before issuing the READ SECTOR command, the Track Register *must* contain the current track that the selected drive is positioned over. If it does not, the Sector Not Found bit will be set when the operation terminates.

If you don't know what track the head is over, you can perform a READ ADDRESS and get the track from that. Then load it into the Track Register.

Figure 4.10

READ SECTOR Routine

```

*****
**                               Example of Reading a Sector.                               **
*****

START  LD      A,1                ;For Drive 0
        LD      (37E0H),A        ;Select
        LD      BC,BUFFER        ;256 Byte Buffer to Store
                                      ; Data That is Read from Disk
        CALL    BUSY             ;Make Sure Controller is Not
                                      ; Busy
        LD      A,88H            ;Read Address Command
        LD      (37ECH),A        ;Issue Command
        PUSH    HL               ;Wait for Controller
        POP     HL
        PUSH    HL
        POP     HL

*****
**                               Data Request Test Loop                               **
*****

LOOP   LD      A,(37ECH) 13       ;Get Stat
        BIT     1,A              12      ;Check Data Request
        JR      NZ,GET  17→12      ;Go if Byte Ready
        RRCA                    4      ;Put Busy in Carry Flag
        RET     NC              15→11  ;Ret if Not Busy, Read Done
        JR      LOOP           12      ;Loop Back

*****
**                               Data Transfer Routine                               **
*****

GET     LD      A,(37EFH) 13       ;Get Byte from Data Reg
        LD      (BC),A           7      ;Store in Buffer
        INC     BC               6      ;Bump Buffer Pointer
        JR      LOOP           12      ;Go Test Again

*****
**                               Busy Test Loop                               **
*****

BUSY    LD      A,(37ECH) 13       ;Get Stat
        RRCA                    4      ;Shift Busy into Carry
        RET     NC              15→11  ;Ret if Not Busy
        JR      BUSY           12      ;Loop Till Not Busy

```

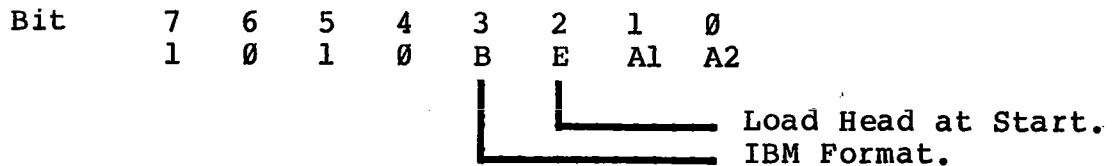
Write Sector Command

This is the command issued before writing a sector. If the sector is not found within two revolutions of the diskette, the Sector Not Found bit is set, and the operation is terminated.

As with the sector read, before issuing the WRITE SECTOR command, the track register *must* contain the current track that the selected drive is positioned over. If it does not, the Sector Not Found bit will be set when the operation terminates.

As previously stated, if you do not know what track the head is over, you can perform a READ ADDRESS and get the track from that. Then load it into the Track Register. This is the format for the WRITE SECTOR command.

Figure 4.11 *WRITE SECTOR Format*



As in all TRS-80 Model I disk operations, the Load Head At Start bit can be ignored because the head is loaded when a drive gets selected.

The IBM Format bit will be discussed in the section on formatting. This bit is normally set during this operation. The A1 and A2 bits are for record type. When writing a sector, sometimes it is desirable to give it some sort of attribute that the software could test to determine if the sector is 'special.' This is done on the DOS's directories to let DOS know these sectors are special-purpose sectors. Your controller does not care what you use your special sectors for. Technically, the attribute is called Record Type. The way these bits (0 and 1) are set in the command byte (when you issue the Write Sector command) will be duplicated in bits 5 and 6 of the Status Register after you read the sector. All your software has to do is test these bits for whatever condition you want. Pretty slick, huh? TRSDOS, NEWDOS and VTOS all use bit 5 (A2) for their Read Protect status.

Figure 4.12 *WRITE SECTOR Routine*

```

*****
**                               Write a Sector Routine                               **
*****

START  LD      A,1                ;For Drive 0
        LD      (37E0H),A        ;Select
        LD      BC,BUFFER        ;256 Byte Buffer That has
                                   ; Data to be Written
        CALL    BUSY             ;Make Sure Controller is Not
                                   ; Busy
        LD      A,0A8H           ;Write Sector Command
        LD      (37ECH),A        ;Issue Command
        PUSH    HL               ;Wait for Controller to Set Up
        POP     HL
        PUSH    HL
        POP     HL

*****
**                               Data Request/Busy Test Loop                               **
*****

LOOP   LD      A,(37ECH)         ;Get Stat
        BIT     1,A              ;Check Data Request
        JR      NZ,PUT           ;Go if Controller Wants a
; Byte
        RRCA                    ;Put Busy in Carry Flag
        RET     NC               ;Ret if Not Busy, Read Done
        JR      LOOP            ;Loop Back

*****
**                               Data Transfer Routine                               **
*****

GET    LD      A,(BC)            ;Get Byte from Buffer
        LD      (37EFH),A        ;Transfer to Controller
        INC     BC               ;Bump Buffer Pointer
        JR      LOOP            ;Go Test Again

*****
**                               Busy Test Loop                               **
*****

BUSY   LD      A,(37ECH)         ;Get Stat
        RRCA                    ;Shift Busy into Carry
        RET     NC               ;Ret if Not Busy
        JR      BUSY            ;Loop Till Not Busy

```

Model III Supplement To Chapter 4

The FDC data I/O commands are the same for the Model III, except that ports are used (remember?), and the NMI and wait state options are used. Single-density disk I/O on the Model III doesn't require waits, but it doesn't hurt.... This means you could have a disk I/O routine that handles single- and double-density. The program in the supplement to Chapter 5 has just this sort of routine.

The NMI Post I/O Processing Routine

Since we are going to use the NMI option, we need a routine to use for a demonstration. The NMI vector at **4049** must contain a jump to this routine before the examples in this chapter can be used. Also, turn off the NMI when doing head movement; it's unnecessary. Turn on the NMI option just before doing I/O.

Store the jump instruction at **4049-404B**.

Figure 4.13 *Vector Jump Routine*

```

XOR      A                ;A = 0
OUT      (0E4H),A         ;Turn NMI Off
LD       A,0C3H           ;JP Opcode
LD       (4049H),A        ;Store Jump
LD       HL,NMIRTN        ;NMI Routine Address
LD       (404AH),HL       ;Store NMI Routine Address

```

Below is the NMI routine.

Figure 4.14 *NMI Routine*

```

NMIRTN   EQU      $
POP      HL                ;Kill Ret Adr (See Text)
IN       A,(0F0H)          ;Get FDC Status
LD       E,A              ;E = FDC Status
IN       A,(0E4H)          ;Get NMI Status
PUSH     AF                ;Save on Stack
XOR      A                ;A = 0
OUT      (0E4H),A         ;NMI Options Off
POP      AF                ;Get NMI Status
AND      64                ;Drives Time Out? Bit 6
RET                          ;Ret with NZ = No Time Out
                                ;Z = Time Out Occured
                                ;E Register = FDC Status

```

Notice the POP HL at the beginning? When a non-maskable interrupt occurs, it's just like forcing a CALL to location 66. So, naturally, the RET address of the I/O loop is still on the stack. The instruction at 66 jumps to another ROM routine which checks for the reset button being pressed. If the reset button is not pressed, it jumps to 4049, where our NMI routine vector is located. We don't want to return to the I/O loop that was interrupted. We want to return to the routine caller, so we simply pop off the loop return address. This makes the caller's return address our return address.

The Model III Read Address Command

The read command is identical to the Model I read command; however, we handle the I/O differently.

Below is an example READ ADDRESS routine. Call with C = select code for drive, i.e. 1, 2, 4 or 8 for drives 0, 1, 2 and 3 respectively. This routine assumes the desired drive is already selected.

Figure 4.15 READ ADDRESS Routine

```

START  LD      A,0C0H          ;A = NMI Options
        OUT     (0E4H),A       ;NMI Options On
        LD      A,C           ;Get Drive Select Code
        OR      128+64        ;Add Double-dens Select and
        LD      D,A           ; Z-80 Wait Select
        LD      HL,BUFFER     ;HL-> Buffer to Receive Data
                                ; Sector ID
        LD      C,0F3H        ;C-> Data Register
        LD      E,2           ;Data Request Mask
        XOR     A             ;A = 0
        OUT     (0E0H),A      ;Disable Normal Interrupts
        DI
        LD      A,0C0H        ;Read Address Command
        OUT     (0F0H),A      ;Issue Command
        CALL    DELAY         ;Delay a Few Microseconds
INIT    IN      A,(0F0H)       ;Get FDC Status
        AND     E             ;Data Ready?
        JR      Z,INIT        ;No - Loop
        INI     A,D           ;Move Data to Buffer
LOOP    LD      A,D           ;Select Drive, Dden and Wait
        OUT     (0F4H),A      ; See Text
        INI     A,D           ;Move Data to Buffer
        JR      LOOP         ;Loop Till NMI Interrupts
DELAY  EX      (SP),HL        ;Wait for FDC to Respond
        EX      (SP),HL
        EX      (SP),HL
        EX      (SP),HL
        RET

```

When the drive is selected at LOOP, with the wait-state option selected, the Z-80 freezes at that point until data is ready. The reason we used the INIT loop is that if no data were found (no disk in the drive) the Z-80 wait at LOOP would re-start after 1.024 milliseconds and erroneous data would be transferred from the FDC.

The Model III Read Track Command

The READ ADDRESS routine in the previous section can be used with the READ TRACK command; just change the command byte from a 0C0 to a 0E0. It will work just fine.

The Model III Write Track (Format) Command

The READ ADDRESS routine can also be used for the WRITE TRACK function. Just change the command byte to an 0F0, and change the INI opcodes to OUTI opcodes. The buffer that HL points to must have the entire track format laid out.

The Model III Read Sector Command

The READ ADDRESS routine can be used for the READ SECTOR function *if* the head is on the desired track, and the track and sector registers contain the desired sector. Use 80 as the command byte.

The Model III Write Sector Command

The READ ADDRESS routine can be used for the WRITE SECTOR function *if* the head is on the desired track, and the track and sector registers contain the desired sector. Use 0A0 as the command byte for normal sectors, and 0A1 for READ PROTECT sectors. Also, substitute the INI opcodes for OUTI opcodes.

5

DISKIO - Full Sector I/O Routine

This chapter contains a full sector (IBM format) I/O driver, and it is ***Bomb-proof***. It will **Never** hang-up for any reason. Feel free to use this routine in any programs that you use or distribute.

It will handle head positioning with full error-checking. The error codes returned are TRSDOS-compatible error codes, except for error 39, which means 'Drive Not Ready.' You can key this into your editor/assembler for use in your programs. By carefully studying each aspect of the driver, you'll gain a more complete understanding of the operations involved.

The write a sector entry is at line 00630 at the WRITE label. The write a protected sector routine starts at line 00610 at the PROT label. Read a sector starts at line 00670 and has the label of READ.

These routines preserve all registers, so you won't have to save them yourself. The register format before calling is as follows:

Figure 5.1 *Full Sector I/O Driver Routine*

ENTRY:

L = Drive number 0-3.
 E = Sector.
 D = Track.
 BC = The 256-byte I/O buffer that data will
 be written to or read from.

EXIT:

All registers the same, except that 'A'
 contains the error code. Zero flag = no error.

Listing Continued...

... Continued Listing

00260	DCT	EQU	\$;DRIVE CONTROL TABLES
00265				;SEE TEXT AFTER DRIVER
00270				;
00280		DEFB	0	;DRIVE NUMBER 0
00290		DEFB	1	;SELECT CODE
00300		DEFB	0	;DRIVE STATUS, BIT REC
00310		DEFB	3	;DRIVE STEP RATE
00320		DEFB	0	;CURRENT TRACK
00330				;
00340		DEFB	1	;DRIVE # 1
00350		DEFB	2	;SELECT CODE
00360		DEFB	0	;DRIVE STATUS
00370		DEFB	3	;DRIVE STEP RATE
00380		DEFB	0	;CURRENT TRACK
00390				;
00400		DEFB	2	;DRIVE # 2
00410		DEFB	4	;SELECT CODE
00420		DEFB	0	;DRIVE STATUS
00430		DEFB	3	;HEAD STEP RATE
00440		DEFB	0	;CURRENT TRACK
00450				;
00460		DEFB	3	;DRIVE # 3
00470		DEFB	8	;SELECT CODE
00480		DEFB	0	;DRIVE STATUS
00490		DEFB	3	;HEAD STEP RATE
00500		DEFB	0	;CURRENT TRACK
00510				;
00520	BUFF	DEFW	0	;I/O BUFFER
00530	TSQ	DEFW	0	;TRK & SECTOR
00540	TRIES	NOP		;COUNTER FOR COUNTING
00550				;# OF TIMES TO TRY I/O
00560				;AFTER ERROR OCCURS
00570	SAVER	DEFW	0	;USED TO STOR HL TEMP
00580				;BY REGS SAVE ROUTINE
00590	TRYS	EQU	3	;# OF TIMES TO TRY I/O
00600				;UNTIL ROUTINE GIVES UP

*** FUNCTION ENTRIES ***

*** WRITE A PROTECTED (DIR TYPE) SECTOR **

00610	PROT	LD	A,0A9H	;FDC COMMAND
00620		JR	WR1	;CONTINUE

*** WRITE A NORMAL SECTOR ***

00630	WRITE	LD	A,0A8H	;FDC COMMAND
00640	WR1	PUSH	HL	;SAVE DRIVE (L)
00650		LD	HL,0012H	;OPCODES: LD A, (BC)
00655				; & LD (DE),A
00660		JR	TASK	

*** READ A SECTOR ***

Listing Continued . .

... Continued Listing

```

00670 READ EQU $
00680 LD A,88H ;READ CMD
00690 PUSH HL ;SAVE DRIVE/OP
00700 LD HL,021AH ;OPCODES: LD A,(DE)
00710 ;& LD (BC),A

```

*** DOMINANT CONTROLLER ***

```

00710 TASK LD (XFER),HL ;XFER Z-80 OPCODES
00720 LD (BUFF),BC ;SAVE I/O BUFF POINTER
00730 LD (TSQ),DE ;SAVE TRACK, SECTOR #'S
00740 LD (ISSUE+1),A ;STORE FDC COMMAND
00750 POP HL ;GET DRIVE NUMBER
00760 CALL FPUSHX ;SAVE ALL REGISTERS
00770 LD A,TRYS ;GET # ERR TRIES
00780 LD (TRIES),A ;STORE IN COUNTER
00790 LD A,0D0H ;FORCE INTERRUPT CMD
00800 LD (37ECH),A ;RESET FDC
00810 CALL GETDCT ;GET CONTROL TABLE PTR
00820 JR Q31 ;CONT

```

*** PUT DRIVE CONTROL TABLE POINTER IN 'IY' ***

```

00830 GETDCT EQU $
00840 PUSH BC ;SAVE REGISTERS
00850 PUSH HL ;BC & HL
00860 LD B,L ;GET DRIVE #
00870 LD HL,DCT ;GET DRIVE TABLE START
00880 ZZ4 INC B ;IS B ZERO YET?
00890 DEC B ;SET Z IF SO
00900 JR NZ,ZYX ;NO, NOT DONE
00910 PUSH HL ;XFER DCT PTR TO REG IY
00920 POP IY ;GET DCT
00930 POP HL ;RESTORE REGS HL
00940 POP BC ; AND BC
00950 RET ;RETURN
;
00960 ZYX LD DE,5 ;ADD FOR NEXT TABLE
00970 ADD HL,DE ;BUMP TO NEXT DCT
00980 DEC B ;DEC DRIVB NUMBER COUNT
00990 JR ZZ4 ;CONTINUE
01000 ;
01010 Q31 CALL TASK1 ;DO WRITE OR READ
01020 LD (HL),0D0H ;RESET FDC
01030 LD A,E ;GET ERROR CODE
01040 OR A ;SET Z OR NZ FLAG
01050 RET ;RETURN TO CALLER

```

*** ACTUAL I/O HANDLER ***

```

01060 TASK1 LD HL,37ECH ;FDC COMMAND/STATUS REG
01070 CALL SELECT ;SELECT DRIVE
01080 BIT 6,(IY+2) ;IS DRIVE DISABLED?
01090 JR Z,TH3 ;NO, CONTINUE
01100 NRER LD E,27H ;NOT READY ERROR CODE

```

Listing Continued...

... Continued Listing

```

01110      DEFB      1      ②  ④ BC, 0F1Eu ;PROT REG 'E'
01120 WPD      LD      E,15 ;WRITE PROTECT ERROR
01130      RET                      ;RETURN
;
01140 TH3      BIT      0,(IY+2) ;IS DRIVE INITIALIZED?
01150      JR      NZ,TH5 ;YES, SKIP INIT PROCESS
01160      LD      A,60H ;'STEP OUT' COMMAND
01170      OR      (IY+3) ;OR WITH STEP RATE
01180      LD      B,80 ;MAX TRACKS POSSIBLE
01190 FGN      CALL     SELECT ;SELECT DRIVE
01200      LD      (HL),A ;ISSUE STEP OUT CMD
01210      CALL     WAIT ;WAIT FOR DONE
01220      BIT      2,(HL) ;DRIVE AT TRACK 0 YET?
01230      JR      NZ,II5 ;YES. INIT DONE.
01240      DJNZ     FGN ;ELSE CONT STEP OUT
01250      JR      NRER ;ERROR IF DIDN'T REACH
01252                      ;TRACK 0 AFTER 80 STEPS
;
01260 II5      SET      0,(IY+2) ;SET INIT BIT IN DCT
01270      LD      (IY+4),0 ;MAKE CURRENT TRACK=0
;
01280 TH5      LD      A,(ISSUE+1) ;GET TYPE, READ/WRITE.
01290      CP      88H ;IS IT READ SECTOR?
01300      JR      Z,TH6 ;GO IF READ
01310      BIT      6,(HL) ;DRIVE WRITE PROTECTED?
01311                      ;WITH TAB OVER SLOT?
01320      JR      NZ,WPD ;YES, GO TO ERROR
01330      BIT      4,(IY+2) ;DRIVE LOGICALLY PROT?
01340      JR      NZ,WPD ;YES, GO TO ERROR
01350 ;
01360 TH6      LD      DE,(TSQ) ;D=TRACK, E=SECTOR
01370      LD      (37EEH),DE ;STORE IN FDC REGS
01380      LD      (HL),0D0H ;RESET FDC
01390      LD      A,(IY+4) ;CURRENT TRACK FROM DCT
01400      LD      (37EDH),A ;FDC TRACK REGISTER
01410      CALL     SELECT ;SELECT DRIVE
01420      LD      A,10H ;SEEK CMD
01430      OR      (IY+3) ;ADD STEP RATE BITS
01440      LD      (HL),A ;ISSUE SEEK COMMAND
01450      CALL     DELAY ;DELAY
01460 JPZ      CALL     SELECT ;SELECT DRIVE
01470      BIT      0,(HL) ;SEEK DONE?
01480      JR      NZ,JPZ ;NO, LOOP
01490 VBX      LD      A,(37EDH) ;GET CURRENT TRACK
01500      LD      (IY+4),A ;STORE IN DRIVE'S DCT
;
01510 REDO      EQU      $
01520      LD      (HL),0D0H ;RESET FDC
01530      LD      BC,(BUFF) ;GET BUFFER POINTER
01540 REOL      LD      DE,37EFH ;DATA REGISTER
01550      CALL     SELECT ;SELECT DRIVE
01560      DI                      ;DISABLE INTERRUPTS
01570 ISSUE     LD      (HL),0 ;ISSUE COMMAND PUT HERE
01580      CALL     DELAY ;DELAY BEFORE TESTING
01590 IOLOO     LD      A,(BC) ;GET BUFFER CHR (WRITES)

```

Listing Continued...

... Continued Listing

01600		JR	DIO2	;TEST FOR DATA REQUEST
01610	XFR	EQU	\$	
01620	XFER	LD	(DE),A	;WRITE CHAR
01630		NOP		;NOP ON WRITES,
01635				;LD (BC),A ON READS
01640		INC	BC	;BUMP BUFFER POINTER
01650		LD	A,(BC)	;GET NEXT BUFFER CHR
01660	DIO2	BIT	1,(HL)	;DATA REQUEST?
01670		JR	NZ,XFR	;YES, GET/PUT BYTE
01680		BIT	0,(HL)	;FULL SECTOR XFERED?
01690		JR	Z,ENDP	;YES, GO END PROCESS
01700		BIT	1,(HL)	;DATA REQUEST?
01710		JR	NZ,XFR	;YES, TRANSFER BYTE
01720		BIT	7,(HL)	;DRIVE NOT READY?
01730		JR	Z,DIO2	;NO, LOOP
01740	ENDP	LD	A,(HL)	;GET FDC STATUS
01750		EI		;ENABLE INTERRUPTS
01760		LD	B,A	;SAVE ERROR CODE
01770	RNAL	RLCA		;DRIVE NOT READY?
01780		JP	C,NRER	;YES, GO TO ERROR
01790		RLCA		;WRITE PROT (WRITE),
01795				;SEC PROT FLAG? (READ)
01800		JR	C,ERROR	;YES, DON'T RETRY I/O
01810		RLCA		;HARDWARE FAULT (WRITE)
01815				;SECTOR IS PROT? (READ)
01820		JR	C,ERROR	;YES, DON'T RETRY I/O
01830		LD	A,B	;GET FDC STATUS AGAIN
01840		LD	E,0	;CLEAR ERROR REGISTER
01850		AND	1CH	;ANY ERRORS?
01860		RET	Z	;NO, RETURN
01870	RNALY	LD	HL,TRIES	;GET TRIES COUNTER
01880		DEC	(HL)	;DEC VALUE
01890		LD	A,B	;GET ERROR CODE
01900		JR	Z,ERROR	;GO IF 'TRIES' EXHAUSTED
01910		RES	0,(IY+2)	;RESET INIT BIT IN DCT 01915
01920		JP	TASK1	;TRY AGAIN, RESEEK
01930	ERROR	LD	A,(ISSUE+1)	;GET FUNCTION CMD
01940		CP	88H	;WAS IT READ?
01950		LD	A,B	;GET ERROR STATUS
01960		JR	NZ,ERR2	;GO IF WRITE
01970		LD	E,3	;READ ERROR CODE START
01980		DEFB	1	;PROT E (LD BC,XX) \neq
01990	ERR2	LD	E,11	;WRITE ERROR START
02000		RRCA		;INIT ERROR SHIFT
02010		RRCA		
02020		RRCA		
02030		RET	C	;RET IF LOST DATA
02040		INC	E	;BUMP ERROR REG
02050		RRCA		;PARITY ERROR
02060		RET	C	;YES, RETURN
02070		INC	E	;BUMP ERROR REG
02080		RRCA		;SECTOR NOT FOUND?
02090		RET	C	;YES, RETURN

Listing Continued...

... Continued Listing

```

02100      INC      E      ;BUMP ERROR REG
02110      RRCA      ;WRITE FAULT (WRITE), 02115
;PROTECT SECTOR? (READ)
02120      RET      C      ;YES, RETURN
02130      INC      E      ;WRITE PROTECT DISK 02133
;OR PROTECTED SECTOR 02140      RET      ;RETURN
WITH ERROR

```

*** SELECT CURRENT DRIVE ***

```

02150 SELECT  PUSH    AF      ;SAVE REG A
02160      LD      A,(HL)    ;GET FDC STATUS
02170      PUSH    AF      ;SAVE STATUS FIRST
02180      LD      A,(IY+1)  ;GET DRIVE SELECT CODE
02190      LD      (37E0H),A ;SELECT DRIVE
02200      POP     AF      ;GET OLD FDC STATUS
02210      RLCA      ;DRIVE ROTATING ALREADY?
02220      JR      C,JKQ    ;GO IF NOT
02230 JJJ     POP     AF      ;RESTORE A
02240      RET      ;RETURN

02250 JKQ     PUSH    BC      ;SAVE BC
02260      LD      BC,8C60H  ;1 SEC DELAY VALUE
02270      BIT     5,(IY+2)  ;WAIT FULL SECOND?
02280      JR      NZ,XPI    ;YES
02290      LD      B,46H     ;ELSE WAIT 1/2 SECOND
02300 XPI     CALL    60H     ;DELAY
02310      POP     BC      ;RESTORE BC
02320      POP     AF      ;AND AF
02330      JR      SELECT   ;RESELECT & RETURN

```

*** DELAY AFTER ISSUING FDC COMMAND **
 *** TO ALLOW FDC TO RESPOND **

```

02340 DELAY  EX      (SP),IX  ;DELAY
02350      EX      (SP),IX  ;ABOUT 30 MICRO SECONDS
02360      RET      ;RETURN

```

*** LOOPS UNTIL FDC IS 'NOT BUSY' ***

```

02370 WAIT   CALL    DELAY    ;DELAY FOR FDC
02380 WA2    BIT     0,(HL)   ;IS FDC BUSY?
02390      RET     Z      ;NO, RETURN
02400      JR      WA2     ;LOOP TIL NOT BUSY
02410      ;

```

*** SAVES ALL REGISTERS ON STACK ***

```

02420 FPUSHX  LD      (SAVER),HL ;STORE TEMPORARILY
02430      EX      (SP),HL    ;GET RETURN ADDRESS
02440      LD      (FPVEC+1),HL ;STORE RET ADR
02450      PUSH    DE      ;SAVE DE
02460      PUSH    BC      ;BC
02470      PUSH    IY      ;IY
02480      PUSH    IX      ;IX
02490      EXX      ;EXCHANGE REGS

```

Listing Continued...

... Continued Listing

```

02500      PUSH    HL          ;HL'
02510      PUSH    DE          ;DE'
02520      PUSH    BC          ;BC'
02530      EXX                ;EXCHANGE BACK
02540      LD      HL, FPOP     ;RETURN ADR FROM ROUTINE
02550      PUSH    HL          ;SAVE ON STACK
02560      LD      HL, (SAVER)  ;RESTORE HL'S VALUE
02570  FPVEC  JP      0         ;RETURN

```

*** HANDLES DE-STACKING REGISTERS ***

```

02580  FPOP  EXX                ;SWITCH REGISTER SET
02590      POP     BC          ;RESTORE BC'
02600      POP     DE          ;DE'
02610      POP     HL          ;HL'
02620      EXX                ;SWITCH REGISTER SET
02630      POP     IX          ;RESTORE IX
02640      POP     IY          ;IY
02650      POP     BC          ;BC
02660      POP     DE          ;DE
02670      POP     HL          ;HL
02680      RET                 ;RETURN

```

The Drive Control Table

Each of the four drives has its own 5-byte 'drive control table,' or DCT for short. The DCT contains information used by the controller in deciding certain conditions.

DCT — Byte 0

This byte contains the drive number (0-3) for the drive to which the DCT belongs. The DISKIO routine doesn't need this byte, but it's handy if you need to write a disk utility and you wish to know which drive the DCT pointed to by the IY Register is using. You can use the GETDCT routine independently of the DISKIO driver in order to get a DCT for your own purposes.

DCT — Byte 1

This byte is the 'binary select code' used in selecting the drive. This value will be 1, 2, 4 or 8 for drives 0, 1, 2 or 3 respectively. You can 'fake out' the DISKIO routine by putting a different select code in this byte so that it really accesses another drive instead.

DCT — Byte 2

This byte contains a bit record of certain drive status information.

Bit 7 means the drive has been 'initialized.' Initialization simply means that a Head Restore to drive 0 was done, in order to maintain the correct track position value.

Bit 6 means the drive is 'logically' write-protected. This is an option that you may set. If this bit is set, DISKIO will reject any writes to the drive just as though the disk in the drive had a write protect tab on it.

Bit 5 is unused.

Bit 4, if set, means the driver is 'logically' turned off. You may set this option. If this bit is set, DISKIO will perform no sector reads or writes.

Bit 3, when set, means DISKIO will delay 1 second after a 'dry' select. If this bit is off, a 1/2 second delay is done.

Bits 2, 1 and 0 are unused.

The I/O RE-TRIES Value

The 'TRYs' value that is set to EQU 3 is the maximum number of I/O re-tries that DISKIO will perform after an error is encountered. DISKIO restores the drive head before each re-try. For Write Protect, Protected Sector or Write Fault errors, no re-trying is attempted.

Explanations of I/O Errors.

I'm sure you've seen all the disk-related errors, but I'll explain what the errors really mean, as well as their usual causes.

CRC means 'Cyclic Redundancy Check, parity,' or 'Checksum.' Translated into plain English, this means that when the controller writes a sector, it does a mathematical calculation, using all the bytes written and comes up with a two-byte number. The controller stores these two values right after the sector. Every time the sector is read, it recalculates the CRC numbers and compares them with the ones written on the disk. If there is a match, the controller considers the read successful and doesn't set the CRC error bit in the Status Register. If the values don't match, the controller considers the read unsuccessful and sets the CRC error bit. Usually a CRC error is caused by getting the disk close to a magnetic field, which changes some of the bits on the sector. Also, using disks that were formatted or written to on one drive might cause parity errors when used on another drive whose head is not aligned just right, thus causing a bad read of the magnetic pulses by the misaligned drive.

LOST DATA is the result of software not keeping up with Data Request on I/O operations. On reads, if the software does not read the current byte in the Data Register before the next byte is ready to be put into the Data Register, the lost data bit is set in the Status Register. During write operations, Lost Data means the software did not furnish a byte fast enough when the DRQ bit was set. The controller writes a byte of 0 in place of the byte that the software was supposed to furnish.

The SECTOR NOT FOUND bit is set when the desired sector was not found, or the track byte in the sector ID did not match the value in the Track Register. This bit will be set after two revolutions of the disk if the desired sector wasn't found.

The WRITE FAULT bit is set when there is a problem between the drive and the controller.

Single-byte I/O versus Sector I/O

Sector I/O is, of course, reading and writing data a sector at a time. Single-byte I/O is a method that allows other routines to read or write a byte at a time without worrying about handling the actual sector I/O. A byte I/O routine would be called in the same manner as the ROM keyboard scan routine (which inputs a byte) or the display and printer routines (which output a byte). The way a byte I/O routine would work is that the routine would maintain a Current Byte in Sector (CBS) pointer, which is incremented every time the routine is accessed. On read operations, the routine would read the next sector (whatever that happens to be), reset its CBS pointer, read the current byte, increment the CBS, and return the byte to the caller. This will continue until the CBS is greater than the sector length. Then the sequence starts again.

Here is a DOS Boot-loader routine that loads in a machine-language file starting at track 0, sector 5 (usual SYS0/SYS), using byte I/O. In the sector read routine, register 'E' contains the sector, and 'D' contains the track of the sector to read. The loader codes used in the loading routine are standard TRSDOS object file loader codes. These codes are interpreted as shown below.

01 NN LL MM, Load at the address MMLL, the next NN bytes (minus two for MM and LL).

02 02 LL MM, Transfer control to MMLL. This is called the transfer address, usually used for the end of file marker (EOF).

05 NN, Skip the following NN bytes. These are header bytes that don't get loaded (REM function).

Figure 5.2 *DOS Boot-loader Routine*

```
*****
**                                     Boot loader routine                                     **
*****

00140      CALL 1C9H                  ;ROM clear screen routine
00150      LD SP,42FFH                ;Init stack to this location
00160      LD DE,0005H                ;Sector = 5, track = 0,
                                      ;starting sector of SYS0/SYS
00170 J1    LD L,0                    ;Zero byte counter
00180      EXX                       ;Save regs for sector reader
```

Listing Continued . . .

Single-byte I/O Versus Sector I/O

... Continued Listing

```
*****
**          Start of Byte Accesssing          **
*****
```

```
00190      CALL BYTE          ;Read next byte from file.
00200      DEC A              ;Is it a LOAD code?
00210      JR NZ,XFER        ;Go if not
00220      CALL BYTE          ;Get number of byte to load+2
00230      SUB 2              ;Get true amt of bytes to load
00240      LD B,A             ;Put in byte counter.
00250      CALL BYTE          ;Get LSB of load address.
00260      LD L,A             ;Put in L
00270      CALL BYTE          ;Get MSB of load address
00280      LD H,A             ;HL now contains load address
00290  R1    CALL BYTE        ;Get SYS0/SYS data byte
00300      LD (HL),A          ;Transfer to memory
00310      INC HL             ;Bump to next load location
00320      DJNZ R1            ;Do until B = 0
00330      JR RUN             ;Go get next loader code
00350  XFER  DEC A            ;Is code the transfer address
                                ;code? (if byte = 2 when read)
00360      JR NZ,IGNORE      ;Go if not. Must be an ignore
                                ;Data code
00370      CALL BYTE          ;Get next transfer code. (2)
00371      CP 2               ;Is it a 2?
00372      JR Z,XFRL          ;Go if it was
00374      LD HL,M2           ;Get no system msg
00375      JP PRINT           ;Display error
00380  XFRL  CALL BYTE        ;Get LSB of transfer address
00390      LD L,A             ;Put in L
00400      CALL BYTE          ;Get MSB of transfer address
00410      LD H,A             ;HL now contains transfer adr
00420      JP (HL)            ;Jump to SYS0
00430  IGNORE CALL BYTE      ;Get # of bytes to read and
                                ;Ignore
00440      LD B,A             ;Put in byte counter
00450      LD HL,0             ;Position over ROM
00460      JR R1              ;Load ignore byte over ROM
                                ;Which does nothing
```

```
*****
**          Routine to Fetch Next Byte          **
*****
```

```
00470  BYTE  EXX              ;Get REGS
00480      LD A,L             ;Get byte count
00490      OR A                ;Set z flag if L=0
00500      CALL Z,GETS         ;Get next sector is L=0
00510      LD A,(HL)           ;Get next byte
00520      INC HL              ;Bump buffer pointer
00530      EXX                ;Save regs for next time
00540      RET                 ;Return
00550  GETS  CALL SECTOR       ;Get next sector
00560      LD HL,4100H          ;Load with input buffer
00570      INC E                ;Bump sector
00580      LD A,E              ;Get next sector name
```

Listing Continued...

... Continued Listing

```

00590      CP      10      ;Time to bump track?
00600      RET C      ;Ret if not
00610      LD E,0      ;Reset to sector 0
00620      INC D      ;Bump to next track
00630      RET      ;Return

```

```

*****
**                               Load Next Sector                               **
*****

```

```

00650      PUSH HL      ;Save register
00660      PUSH BC
00670      LD HL,37ECH      ;Controller command/status reg
00680      LD (HL),0D0H      ;Reset controller
00690      LD A,1      ;Select code for drive 0
00700      LD (37E0H),A      ;Re-select the drive
00710      LD (37EEH),DE      ;Load controller with sector & track
00720      CALL WAIT      ;Let controller react
00730      LD (HL),13H      ;Issue the seek command
00740      CALL BUSY      ;Wait for seek to finish
00750      LD BC,4100H      ;Load with input buffer
00760      LD (HL),88H      ;Issue the sector read command
00770      CALL WAIT      ;Let controller react
00780  LOOP   BIT 1,(HL)      ;Byte in data reg?
00790      JR NZ,GET      ;Go if byte ready
00800      BIT 0,(HL)      ;Read done? (not busy)
00810      JR Z,DONE      ;Go if read done
00820      JR LOOP      ;Loop back
00830  GET    LD A,(37EFH)      ;Read byte for data register
00840      LD (BC),A      ;Store in input buffer
00850      INC BC      ;Bump buffer pointer
00860      JR LOOP      ;Loop back
00880  DONE   LD A,(HL)      ;Get controller status
00890      POP BC      ;Restore registers
00900      POP HL
00910      AND 1CH      ;Strip out errors
00920      RET Z      ;Ret if no error
00930      LD HL,DISKE      ;Get DISK ERROR message

```

```

*****
**                               Message Printer                               **
*****

```

```

00940  PRINT  LD A,(HL)      ;Get message byte
00950      INC HL      ;Bump message pointer
00960      OR A      ;Set Z if A = 0
00970      JR Z,STOP      ;Go if end of message
00980      CALL 33H      ;Display byte
00990      JR PRINT      ;Loop
01000  STOP   CALL 49H      ;Wait for a keyboard input
01002      JP 0DH      ;Jump to BASIC bootstrap loader
01010  BUSY   CALL WAIT      ;Let controller react
01020  B1     BIT 0,(HL)      ;Is controller busy?
01030      RET Z      ;Ret if not busy.
01040      JR B1      ;Loop

```

Listing Continued...

... Continued Listing

```

*****
**                               Controller Delay Routine                               **
*****

01050 WAIT    PUSH HL                ;Lets controller react
01060         POP HL
01070         PUSH HL
01080         POP HL
01090         RET

*****
**                               Disk Error Message Text                               **
*****

01100 DISKE   DEFM 'DISK ERROR' ;Disk error message
01110         NOP                ;Print delimiter

```

Notice in the GETS subroutine that every time 'E' (the sector counter) reached 10, it was reset to zero, and 'D' (the track counter) was bumped up one.

This routine assumes the machine-language program being looked for starts at track 0, sector 5. Also, all of the program's sectors must be consecutive.

This routine is very similar to how the DOS's system file overlay (SYS1, SYS2 etc . . .) loader works, except that the overlay loader checks the directory to see where on the disk an overlay starts.

Model III Supplement To Chapter 5

This supplement contains a Model III disk driver which handles sector reads and writes. It can read and write single or double density, but does not automatically recognize density. Selecting density is done by changing a bit in the Drive Control Table (DCT).

The DCT

The DCT is located on line 00630. There is a DCT for each drive. The DCT contains information needed for efficient disk I/O.

Below is a list of the DCT bytes and their use.

Figure 5.3 DCT Bytes

BYTE	USE
0	7 Drive current density. 1=Double 6-4 Unused 3-0 Drive select code, i.e 0001 = drive 0, 0010 = drive 1, etc.
1	Drives current track.
2	Drive status 7 0=Drive needs head restore. 6-0 Undefined. You may use these.

Byte 0 is simply the drive's select code with the density bit in bit 7. This makes sense, since bit 7 is the bit used in selecting density via port F4.

Byte 1 is the drive's current track. This allows the Track Register to be loaded with the correct value before SEEKing is performed.

Byte 2 is a status byte. The only bit currently used is bit 7. If this bit is off, the drive needs to be INITIALIZED. Initializing a drive consists of restoring the head to track 0 and updating byte 1 of the DCT to 0, then setting bit 7 of DCT byte 2. This allows the disk I/O routine to then know exactly where the head is. This is usually done only once, before the first I/O is attempted for a given drive. Certain disk errors cause the initialization to be done again.

Using The Disk Drive in a Program

To incorporate this disk driver into a program, simply type it into a TRS-80 compatible editor/assembler (such as Radio Shack, Apparat, Assem/80, or EDAS).

Below is the register format for routine entry.

Figure 5.4 Register Entry Format

C = Drive number
 D = Track number
 E = Sector number
 HL -> Buffer for data I/O.

READ SECTOR CALL READ, line 00680
 WRITE SECTOR CALL WRITE, line 00730
 WRITE PROTECTED SECTOR CALL PROT, line 00780

Model III Disk Driver Routine

Figure 5.5 Model III Disk Driver Routine

```

00040 ;          DISKIO, DISK ROUTINES FOR MODEL III
00060 ;
00070 CMD      EQU      0F0H
00080 TRK      EQU      0F1H
00090 SEC      EQU      0F2H
00100 DATA    EQU      0F3H
00110 SEL      EQU      0F4H
00120 INT      EQU      0E0H
00130 NMI      EQU      0E4H
00140 TRIES    EQU      5
00150 ;
00160 TAS      DW        0          ;STORAG
00170 BUFF     DW        0          ;BUFFER STO
00180 TRIC     NOP              ;# I/O ATTEMPTS COUNTER
00190 ;
00200 SELECT   IN        A,(CMD)    ;GET STAT
00210          PUSH     AF          ;SAV STAT
00220          LD       A,(IX)      ;GET CURRENT SEL COD
00230          OUT      (SEL),A     ;SEL
00240          POP      AF          ;GET STAT
00250          RLCA              ;DRIVES SEL?
00260          RET      NC          ;Y
00270          LD       BC,2000H    ;DELAY
00280 S1       DEC      BC          ;DEC COUNT
00290          LD       A,B
00300          OR       C
00310          JR      NZ,S1        ;NOT DONE
00320          JR      SELECT      ;RESEL, RET

```

Listing Continued . . .

... Continued Listing

```

00330 ;
00340 DELAY    EX      (SP),HL
00350          EX      (SP),HL
00360          EX      (SP),HL
00370          EX      (SP),HL
00380          RET
00390 ;
00400 WAIT     CALL    DELAY          ;DELAY
00410 WHO      CALL    SELECT        ;SELECT DRV
00420          IN      A,(CMD)        ;GET STAT
00430          RRCA          ;BUSY?
00440          RET      NC            ;N
00450          JR      WHO            ;LOOP
00460 ;
00470 BUSY      IN      A,(CMD)        ;STAT
00480          RRCA          ;BUSY?
00490          JR      C,BUSY         ;N
00500          RET
00510 ;
00520 GETDCT    LD      A,C            ;GET DRV
00530          RLCA          ;X2
00540          RLCA          ;X4
00550          LD      E,A            ;XFR
00560          LD      D,0
00570          LD      HL,DCT         ;DCT
00580          ADD     HL,DE          ;HL=DRV DCT
00590          PUSH   HL            ;SAV TO IX
00600          POP    IX
00610          RET
00620 ;
00630 DCT       DM      1+128,0,0,3
00640          DM      2+128,0,0,3
00650          DM      4+128,0,0,3
00660          DM      8+128,0,0,3
00670 ;
00680 READ      CALL    TASK          ;JP SET-UP
00690          DB      3              ;LOST DATA READ
00700          DB      80H           ;READ CMD
00710          DB      0A2H         ;INI
00720 ;
00730 WRITE     CALL    TASK          ;JP SET-UP
00740          DB      11             ;LOST DATA, WRITE
00750          DB      0A0H         ;WRITE CMD
00760          DB      0A3H         ;OUTI
00770 ;
00780 PROT      CALL    TASK          ;JP SET-UP
00790          DB      11             ;LOST DATA WRITE
00800          DB      0A1H         ;WRITE PROT CMD
00810          DB      0A3H         ;OUTI
00820 ;
00830 ;          START I/O PROC
00840 ;
00850 TASK       EQU      $
00860          LD      (BUFF),HL        ;SAV BUFF ADR
00870          LD      (TAS),DE         ;STO TK & SEC

```

Listing Continued...

```

00880      LD      A,TRIES      ;# I/O TRIES
00890      LD      (TRIC),A      ;STO IN COUNTER
00900      XOR      A      ;CLR
00910      OUT     (NMI),A      ;NMI OFF
00920      OUT     (INT),A      ;INT OFF
00930      DI      ;INT OFF
00940      EX      (SP),HL      ;PUSH HL
00950      ;HL->I/O CODES
00960      LD      A,(HL)      ;GET FUNC 1ST ERR CODE
00970      LD      (ERR+1),A      ;STO
00980      INC     HL      ;BMP PTR
00990      LD      A,(HL)      ;GET I/O CMD
01000      LD      (IO+1),A      ;STO
01010      INC     HL      ;BMP PT
01020      LD      A,(HL)      ;GET INI, OUTI CODE
01030      LD      (OPCODE+1),A      ;STO OPCODE
01040      LD      (OPCOD2+1),A      ;STO OPCODE
01050      PUSH    DE
01060      PUSH    BC
01070      PUSH    IX
01080      CALL    GETDCT
01090  TS1      CALL    TASK2      ;DO FUNC
01100      LD      A,0D0H      ;RESET
01110      OUT     (CMD),A      ;FDC
01120      LD      A,E      ;GET ERR
01130      OR      A      ;ERR?
01140      JR      Z,NRT      ;N, RET
01150      AND     7      ;RE-TRY ERR?
01160      JR      Z,NRT      ;GO IF ERR 8
01170      CP      6      ;?
01180      JR      NC,NRT      ;N
01190      LD      HL,TRIC      ;TRY COUNTER
01200      DEC     (HL)      ;DEC TRIES
01210      JR      NZ,TS1      ;TRI AGAIN
01220  NRT      LD      A,E      ;GET ERR COD
01230      OR      A      ;SET NZ
01240      POP     IX      ;RES REGS
01250      POP     BC
01260      POP     DE
01270      POP     HL
01280      RET
01290 ;
01300  TASK2      CALL    SELECT      ;SELECT DRIVE
01310      BIT     7,(IX+2)      ;DRV INIT?
01320      JR      NZ,IP      ;Y
01330      LD      A,0D0H      ;RESET
01340      OUT     (CMD),A      ; FDC
01350      LD      B,80      ;MAX TRKS
01360  STP      CALL    SELECT      ;SEL DRV
01370      IN      A,(CMD)      ;GET STAT
01380      BIT     2,A      ;TRK 0?
01390      JR      NZ,STQ      ;Y
01400      LD      A,60H      ;STEP CMD
01410      OR      (IX+3)      ;STEP RATE
01420      OUT     (CMD),A      ;ISSUE STEP

```

Listing Continued...

... Continued Listing

01430		CALL	WAIT	;WAIT FOR STEP
01440		DJNZ	STP	;TEST FOR TRK 0
01450		LD	E,8	;DEV NOT READY
01460		RET		
01470	STQ	LD	(IX+1),0	;STO CURR TRK
01480		SET	7,(IX+2)	;INIT DONE
01490	IP	LD	A,(IX+1)	;GET DRV CUR TRK
01500		OUT	(TRK),A	;STO IN FDC
01510		LD	HL,(TAS)	;GET TK & SEC
01520		LD	A,L	;GET SEC
01530		OUT	(SEC),A	;FDC=SEC
01540		LD	A,H	;GET TRK
01550		OUT	(DATA),A	;STO DESIRED TRK
01560		LD	A,10H	;SEEK CMD
01570		OR	(IX+3)	;ADD STEP RATE
01580		OUT	(CMD),A	;ISSUE SEEK
01590		CALL	WAIT	;LOOP TILL DONE
01600		IN	A,(TRK)	;GET CUR TRK
01610		LD	(IX+1),A	;STO
01620	;			
01630		LD	A,0D0H	;RESET
01640		OUT	(CMD),A	;FDC
01650		LD	HL,HERE	;NMI RTN
01660		LD	(404AH),HL	;STO VEC
01670		LD	A,0C3H	;JP OP
01680		LD	(4049H),A	;STO JP
01690		LD	A,0C0H	;NMI REQ
01700		OUT	(NMI),A	;SEL NMI
01710		LD	A,(TRK)	;GET SEL COD
01720		CP	22	;WAITS
01730		LD	A,(IX)	;GET TRK
01740		SET	6,A	;PRECOMP NEEDED?
01750		JR	C,SAI	;N
01760		OR	32	;ADD PRECOMP
01770	SAI	PUSH	AF	;SAV
01780		LD	E,2	;DRQ MASK
01790		LD	B,0	;BYTE COUNT
01800		LD	C,DATA	;DATA REG
01810		LD	HL,(BUFF)	;GET I/O BUFF
01820		DI		;KILL INT
01830		XOR	A	;A=0
01840		OUT	(INT),A	;NO INT
01850	IO	LD	A,80H	;GET CMD
01860		OUT	(CMD),A	;ISSUE CMD
01870		CALL	DELAY	;DELAY A SEC
01880	LOOP	IN	A,(CMD)	;GET STAT
01890		AND	E	;DRQ?
01900		JR	Z,LOOP	;N
01910	OPCODE	INI		;GET/PUT BYTE
01920		POP	AF	;GET SEL COD
01930	A00	OUT	(0F4H),A	;SEL, WAIT
01940	OPCOD2	INI		;MOV DATA
01950		JR	A00	;LOOP
02000	;			
02070	HERE	EQU	\$	

Listing Continued...

... Continued Listing

02080	POP	HL	;KILL LOOP RET
02090	IN	A, (CMD)	;GET DISK STAT
02100	PUSH	AF	;SAV STAT
02110	IN	A, (NMI)	;GET NMI STAT
02120	PUSH	AF	;SAVE NMI STAT
02130	XOR	A	;A=0
02140	OUT	(NMI), A,	;NMI OFF
02150	POP	AF	;GET NMI STAT
02160	AND	64	;DRIVE TIME OUT?
02170	JR	NZ, HR1	;N
02180	LD	E, 8	;DEV NOT AVAIL
02190	RES	7, (IX+2)	;INIT OFF
02200	POP	AF	;GET DISK STAT
02210	RET		
02220	HR1 POP	AF	;GET DISK STAT
02230	LD	E, A	;E=A
02240	OR	A	;NO ERR?
02250	RET	Z	;Y
02260	ERR LD	E, 0	;GET ERR START
02270	RRCA		;SHIFT PAST BUSY
02280	RRCA		; DRQ
02290	RRCA		;LOST DATA?
02300	RET	C	;Y
02310	INC	E	;BMP ERR COD
02320	RRCA		;PARITY ERR?
02330	RET	C	;Y
02340	INC	E	;BMP ERR COD
02350	RRCA		;SEC NOT FOUND?
02360	RET	C	;Y
02370	INC	E	;BMP ERR COD
02380	RRCA		;PROT SEC/WRITE FAULT?
02390	RET	C	;Y
02400	INC	E	;BMP ERR COD, MUST BE
02410			;PROT SEC/WRITE PROT?
02420	RET		

6

Using TRSDOS/NEWDOS/VTOS I/O Routines

In this chapter we will be dealing with using the DOS's file I/O routines and other DOS trivia. Some programs in the back of the book use DOS file I/O.

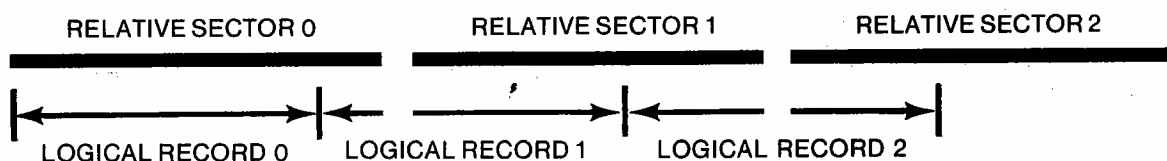
TRSDOS/NEWDOS/VTOS Disk Files

Why a disk operating system? The DOS performs many useful functions. The main function is handling a certain logical contrivance we commonly call 'files.' The DOS handles allocation and de-allocation of disk space for the files. Without the DOS, we would have to take care of this big housekeeping chore ourselves. Maybe after mastering control of the disk, you might attempt to write your own DOS!

There are two methods of accessing disk files: direct record accessing (also referred to as random access) and sequential byte accessing.

Disk BASIC allows you to access files with random file accessing. Any record in the file may be read or written at any time. The DOS will allow logical record lengths (LRL) to be different from the actual physical sector length. This may be from 1 to 256 bytes per record.

The DOS will handle 'spanning' sectors if a particular record is spanned over two sectors. For example, if your LRL was 200 bytes long, the first record would fit on the first sector of the file. The first 55 bytes of the second record would fill up the rest of the first sector, and the remaining bytes of the second record would fill up the first 145 bytes of the second sector, and so on.

Figure 6.1 *Record Spanning Two Sectors*

Byte I/O may also be performed, just like reading or writing a byte from a device.

When speaking in computer terms, do you know what a device is? Your printer, video and keyboard are all devices. Mechanical card puncher/readers and paper tape punch/readers are devices. A good definition for a computer device is any mechanical or logical contrivance capable of supplying and/or receiving/processing data.

When writing to a DOS file in the byte I/O mode, it is appropriate to think of the file as a logical device that is storing all those bytes in one giant buffer, to be read back in or processed at a later time.

A legal TRSDOS/NEWDOS/LDOS file specification (filespec) must include the primary filename, which may be up to 8 characters long. The first character must be a letter from A to Z, although the rest of the characters of the primary filename may be alphanumeric (letters or numbers).

An optional extension may be included to denote the filetype or whatever. This may be up to 3 characters in length, but like the primary filename, the first character must be a letter from A to Z.

An optional 8-character password may be included to prevent unauthorized persons from accessing the file. The character rule is the same as the primary filespec. The first character must be a letter, and the rest of the characters of the password may be alphanumeric.

An optional 2-character drive specifier can be included to specify which drive to use. The first character must be a colon. This denotes the drivespec. The second character is the actual drive number. It must be from 0 to 3. If no drivespec is specified, the DOS will scan all drives for the file and open the first file that matches the filespec. Depending on the call used, the DOS may create the file on the first available drive, or return with a 'file not found error.'

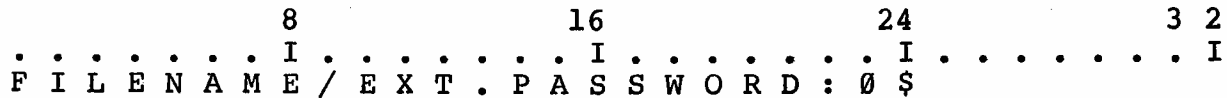
Before calling the open file routine, or before calling any DOS file-handling routine, Register 'DE' must point to the first byte of a 32-byte RAM buffer which contains the filespec. This is called the FILE CONTROL BLOCK (FCB). It is common to hear this referred to as the DEVICE CONTROL BLOCK (DCB).

The filespec must be positioned in the FCB so that all the unused bytes are on the right side of the filespec, or in other words, left set in the buffer. When the file is open, DOS keeps all the file's housekeeping information in this FCB buffer.

Below is a visual description of how the filespec should be set in the FCB prior

to calling the DOS open routines. Each dot represents one byte. The dollar sign (\$) represents the terminator of **03H** or **0DH** that must follow after the last filespec byte. The 'DE' Register must point to the first byte of the FCB before calling any of the file control routines.

Figure 6.2 *Filespec in the FCB*



Opening a New or Existing File

In order to open a new or existing disk file, you must point Register 'DE' to the 32-byte RAM buffer that holds the filespec, and is delimited by a byte, **03** or **0D**, right after the last filespec character.

The logical record length, from 0 to 255 (0 being full 256-byte sector record) must be put in the 'B' Register. If you are just going to access the file in the byte I/O mode, this value has no significance whatsoever.

'HL' must be pointed to the 256-byte I/O buffer, which is used by DOS to transfer data to and from the sectors of the file. This value is stored in the FCB during 'open' and does not need to be supplied again.

After setting up the FCB and the registers, execute a call to location 4420.

i. On return, 'BC', 'HL' and 'DE' are intact. The zero flag will be set if no error occurred; otherwise, Register 'A' will contain the error code. If a new file was created, the carry flag will be set. Error processing will be discussed later.

If a drive was specified, DOS will look for the file on that drive. If the file is not there, DOS will create its entry in the disk's directory, and return.

If a drive was not specified, DOS will scan all the drives starting at drive zero, and search for the first occurrence of the filespec. If DOS finds the file, it will open it and return; otherwise, DOS will create the file on the first drive that contains free directory space and return.

In the FCB, DOS maintains a 3-byte value called the 'EOF' (end of file) byte. The first 2 bytes contain the number of the last sector, and the 3rd byte contains the relative byte position of the last byte in the file.

Every time a byte or record is written, the EOF byte value is changed to equal that record/byte. This is most undesirable when maintaining direct access record files. For example, if you had 10 records, and you wrote to record 2, the EOF would be set back to record 2, and the last 8 would be de-allocated! This circumstance *is* desirable when writing byte output sequential files because you would want any space that wasn't just written to freed.

The way to prevent the EOF value from being reset when records or bytes are written is to turn on bit 6 of byte 2 of the FCB right after you open the file.

Below is an example of a call to open a new or existing file, setting the 'Don't Reset EOF' bit, LRL = 128.

Figure 6.3 New or Existing File Call Routine

INIT@

```
*****
**          OPEN a New or Existing File.          **
*****

START  LD      DE,FCB          ;32 BUFF THAT HOLDS FILESPEC
        LD      HL,BUFFER      ;FILES' I/O BUFFER
        LD      B,128          ;128 BYTE LRL
        CALL    4420H          ;OPEN OR CREATE FILE
        JR      NZ,ERROR       ;GO IF ERROR
        INC     DE             ;POSITION TO 2ND BYTE OF FCB
        LD      A,(DE)         ;GET BYTE
        SET     6,A            ;SET "DON'T RESET EOF" BIT
        LD      (DE),A         ;REPLACE
        RET
ERROR   (The error processing would be executed here).
```

Opening an Existing File

This call enables you to open an existing file, but DOS will not create it if the file was not found. This call would be the call to make if, for example, you wanted to read in a text file in the byte mode, and you didn't want DOS to create the file if it did not find it. You wouldn't want to create an unwanted directory entry just because the desired file was not in the system. The register set up is exactly like the open/create call. To open an existing file, call 4424.

Below is an example of an open call, not setting the 'don't reset EOF' bit. LRL = full 256-byte record.

Figure 6.4 File Call Routine

OPEN@

```
*****
**          OPEN an Existing File Only          **
*****

START  LD      DE,FCB          ;32 BUFF THAT HOLDS FILESPEC
        LD      HL,BUFFER      ;FILES' I/O BUFFER
        LD      B,0            ;256 BYTE RECORD.
        CALL    4424H          ;OPEN FILE.
        RET      Z             ;GO IF NO ERROR
ERROR   (Error processing is done now).
```

Performing Direct Record I/O

After you open a file, DOS keeps a value called the Next Record To Access (NRA) in the file's 32-byte FCB. When the file is first opened, NRA is set to 0, meaning the next record is record 0. Every time a read or write is done, the NRA is incremented by one record.

When accessing a direct record, it is necessary to position the NRA to the record you desire to read using the 'Position to Record' call. After this has been done, you may call the read or write routines.

To position to a desired record, the 'DE' Register must point to the FCB of the open file that is to be accessed, and 'BC' contains the record number to position over. Now call 4442. If an error occurred, its code will be in the 'A' Register. The zero flag will be set if no error occurred.

Figure 6.5 Positioning to a Record

POSN @

```
*****
**          POSITION to a Logical Record          **
*****

LD      DE,FCB          ;32 BYTE FILE CONTROL BLOCK
LD      BC,10           ;POSITION NRA TO RECORD 10
CALL    4442H           ;CALL POSITION ROUTINE.
RET     Z               ;RET IF NO ERROR
Error processing goes here.
```

9

If you call a read or write routine, and your LRL = 256 (B = 0 at open time), then the file's 256-byte I/O buffer is read or written. 'HL' is pointed to the buffer at open time, and it is stored in the FCB.

If you call a read or write routine, and your LRL is not 256, 'HL' must point to *another* buffer called 'Userrec.' Userrec must be the same length as the file's LRL. This buffer is different from the 256-byte I/O buffer defined by 'HL' at open time. DOS will read and write the logical record to and from the Userrec buffer.

To read a record, call 4436. To write a record, call 4439.

Figure 6.6 Reading a LRL Record

READ @

```
*****
**          Example of Reading a LRL=256 Record          **
*****

LD      DE,FCB          ;FILE CONTROL BLOCK
LD      BC,20           ;RECORD 20
CALL    4442H           ;POSITION TO RECORD 20
```

Listing Continued . . .

... Continued Listing

```

JR      NZ,ERROR      ;GO IF ERROR
CALL    4436H          ;READ SECTOR INTO I/O BUFFER
JR      NZ,ERROR      ;GO IF ERROR
RET      ;RET
ERROR    (Error processing goes here.)

```

Figure 6.7 Writing a Record Using Userec

WRITE@

```

*****
** Example of writing a record whose LRL=130. USEREC must **
**          contain data to be written.                  **
*****

```

```

LD      DE,FCB          ;FILE CONTROL BLOCK
LD      BC,12           ;RECORD 12
CALL    4442H          ;POSITION TO RECORD 12
JR      NZ,ERROR      ;GO IF ERROR
LD      HL,USEREC       ;DATA BUFFER
CALL    4436H          ;READ RECORD IN TO USEREC
JR      NZ,ERROR      ;GO IF ERROR
RET      ;RET
ERROR    (Error processing goes here.)

```

If you want to write a record and have DOS verify-read it, call 443C. This increases the time of the operation by about 100 percent, but it may save important data from being lost.

VER@

To 'rewind,' or position the current byte/record counters back to the beginning of the file, call 443F. This does not affect the EOF value unless bit 6 of the 2nd byte of the FCB is off.

REW@

To 'backspace' one record, call 4445. This has no effect on the EOF unless bit 6 of the 2nd byte of the FCB is off.

BKSP@

To position the current record/byte counters to the EOF, call 4448. This is useful in adding to the end of a sequential file.

PEOF@

Performing Single-byte I/O

Remember when we discussed single-byte I/O? DOS gives you the capability of reading or writing single bytes. When files are first opened, the Current Byte Position is set to zero. This is not the EOF value.

Every time a byte is read or written, this value is incremented by one. This is so DOS can keep track of the next byte to read or write. Byte I/O should be logically thought of as reading or writing a byte to a logical device, such as the keyboard or the printer. Files accessed by byte I/O are referred to as *sequential* files, because the file is logically thought of as one long sequence of bytes.

Byte I/O is very useful. For example, that's how Disk BASIC saves and loads its programs, and how the machine-language loader in DOS accesses the bytes of the file. Using byte I/O, one could easily write a driver that would route the bytes going to the printer to a disk file! Or, you could write a keyboard driver that would get all of the bytes from a file instead of the keyboard.

When calling the byte read/write routines, 'DE' must point to a FCB of an open file, and if a write is done, Register 'A' must contain the byte to write. On return, 'A' = byte written or read.

The call to read a file byte is 13, and the call to write a file byte is 1B. Yeah, I know these are ROM addresses, but they jump to a DOS vector. Below is an example of a routine that would dump the contents of all memory from 5300 to 6300 to a currently open disk file.

Figure 6.8 Dumping Memory to an Open File

```
*****
**                               Dump Memory Using Byte Output                               **
*****

START  LD      HL,5300H          ;STARTING ADDRESS
        LD      DE,FCB          ;OPEN FILES' FCB
L1      LD      A,(HL)           ;GET BYTE TO WRITE
        CALL    1BH             ;WRITE TO FILE.
        JR      NZ,ERROR        ;GO IF ERROR
        INC     HL              ;BUMP MEMORY POINTER
        PUSH    HL              ;SAVE MEM POINTER
        PUSH    DE              ;SAVE FCB POINTER
        LD      DE,6301H        ;GET LIMIT+1
        OR      A               ;CLEAR CARRY FLAG
        SBC     HL,DE           ;COMPARE
        POP     DE              ;RESTORE FCB
        POP     HL              ;RESTORE MEM PTR
        RET     Z               ;RET IF DONE
        JR      L1             ;LOOP BACK
ERROR   (Error processing goes here) .
```

Closing Files

If record-writes past the file's current EOF are performed, or the EOF is different from when you opened the file, it must be closed to store the necessary data used by DOS in the directory. It is good practice to close a file if any writes have been done, just in case. Closing is never needed if the file was only read from.

To close the file, load 'DE' with the open file's DCB and call 4428. After the close, DOS puts the filename and extension (if any) back in the FCB buffer. DOS does not put the password back in. 'A' contains any error. The zero flag is set if there was no error. The close routine will de-allocate any grans past the gran that contains the EOF sector.

... Continued Listing

- 16/10 ILLEGAL LOGICAL FILE NUMBER
The FCB contains bad data. 'DE' probably wasn't pointed to the correct FCB.
- 17/11 DIRECTORY READ ERROR
An error occurred when a directory sector was read.
- 18/12 DIRECTORY WRITE ERROR
An error occurred when DOS tried to write a dir sector.
- 19/13 ILLEGAL FILESPEC
The filespec contained illegal characters, was too long, etc.
- 20/14 GAT READ ERROR
An error occurred when DOS read the granule allocation table.
- 21/15 GAT WRITE ERROR
An error occurred when DOS tried to write to the granule allocation table.
- 22/16 HIT READ ERROR
An error occurred when DOS read the hash index table.
- 23/17 HIT WRITE ERROR
An error occurred when DOS wrote to the hash index table.
- 24/18 FILE NOT IN DIRECTORY
The file that you tried to open without creating was not found.
- 25/19 FILE ACCESS DENIED
You tried to access a file beyond the access code assign to the access password.
- 26/1A DIRECTORY SPACE FULL
No more room is left in the directory to create another file
- 27/1B DISK SPACE FULL
There is no available disk space on the disk.
- 28/1C EOF ENCOUNTERED.
The byte/record just accessed was ^(past) the end of file byte/record.
- 29/1D PAST END OF FILE
An attempt to access past the EOF byte/record.
- 30/1E FULL DIRECTORY
This error would occur if a record write was done, and the file entry in the directory needed to be extended, but no directory space was left.

Listing Continued . .

Continued Listing

- 31/1F PROGRAM NOT FOUND
The desired CMD program was not found.
- 32/20 ILLEGAL DRIVE NUMBER
The drive number in the filespec was bad.
- 33/21 DEVICE SPACE FULL - (used in VTOS only)
No more room in the device area left.
- 34/22 LOAD FILE FORMAT ERROR
An attempt was made to execute a non-object file.
- 35/23 BAD RAM MEMORY
DOS detected a bad ram location.
- 36/24 TRIED TO LOAD ROM
An attempt was made to load ROM with object code. I don't know WHY this error exists because you cant hurt ROM by trying to load over it anyhow.
- 37/25 ACCESS TO PROTECTED FILE DENIED
The update password was not supplied, and the access password's access level = no access.
- 38/26 FILE NOT OPEN
The FCB was not an open FCB.

Past EOF Error Messages

When an error occurs while writing a record past EOF, or if EOF is greater than it was before the file was opened, it is a good idea to immediately close the file.

If you've been using one of the DOSes, I'm sure you've seen the error messages. If you want to display a DOS error message, then call 4409. This is the vector for the DOS error message display routine. This overlay will display the error code in Register 'A'. If you want a return from the error displayer, bit 7 of Register 'A' must be set. If it's not, the routine will exit by a jump to 402D, which is the DOSREADY vector. If you want the short error messages, bit 6 of the 'A' Register must be 1; otherwise, extra diagnostics will be displayed as shown. Page 6-12 of the TRSDOS manual is in error when it states that by OR'ing the error code with 80, the diagnostics will be displayed.

ERROR@

EXIT@

Figure 6.12 Error Display Bits

```

7 6 5 4 3 2 1 0
: : 0-5 contain error code.
: :...1 if no diagnostics.
: :.....1 if return desired.

```

Here is an example of a short error message:

LOST DATA DURING READ

Here is an example of an error message with diagnostics:

Figure 6.13 Error Message

```

** ERRCOD=03, LOST DATA DURING READ **
  <FILE=STOCK/DAT>
  REFERENCED AT X'8233'

```

TRSDOS, NEWDOS 2.1 and VTOS all have this diagnostic display format. NEWDOS/80 does not have diagnostics. The diagnostics are not very useful, and my guess is that Randy Cook (original author of TRSDOS) probably used them for debugging TRSDOS and its utilities. Below is an example of a routine that calls the error routine, but does not use the diagnostics, and expects the error routine to return control when done.

Figure 6.14 Error Call Routine

```

*****
**          Routine Using the DOS Error Displayer          **
*****

START  CALL    4436H          ;READ A RECORD
        RET     Z            ;RET IF OK
        OR      C0H          ;SET RET BIT, AND NO DIAGNOS
        CALL    4409H          ;DISPLAY ERROR
        RET                      ;RET

```

Other DOS Functions

Since most of these are not listed in your TRSDOS manual, I have included a list of other useful DOS calls and jumps. These are all applicable to VTOS and NEWDOS/80; however, NEWDOS/80 and VTOS systems have other calls that are not listed here because they are not compatible with TRSDOS.

^{EXIT@ CMD@}
402D or 4400 are the normal program exits to the DOS READY mode.

ABORT@

4030 is the abnormal program exit to DOS READY. If DEBUG is active, it will be invoked.

CHNDI@

4405 will cause the DOS command interpreter to execute the command, delimited with a byte of **0D**, pointed to by the 'HL' Register pair. After the command or program is executed, DOS READY will be invoked via **402D**. One way to make this return to your program would be to load **402E** with the address to which you wanted DOS to jump after executing the command, but you must fix this after it has returned.

ERROR@

4409 is the vector of the previously discussed DOS error display routine.

DEBUG@

440D will jump to the DEBUG monitor. This may also be done by executing an **RST 28** call. The **BREAK** invocation will not be enabled. A return to the program is done by executing a **G 'ENTER'** in DEBUG.

FSPEC@

441C will extract a filespec from a string of text pointed to by the 'HL' Registers, put them in the buffer pointed to by the 'DE' Registers and put a **03** as a terminator after the last byte transferred. **Z** will be set if 'HL' is left pointing to the terminator (if the terminator was **03** or **0D**). Otherwise, 'HL' will be left pointing to the byte after the terminator. (comma)

DSPLY@

4467 will output a line of text pointed to by 'HL' to the video and return. The text is terminated with either a **0D** (which is displayed) or **03** (which is not displayed).

PRINT@

446A will output a line of text pointed to by 'HL' to the printer and return. The text is terminated with either a **0D** (which is printed) or **03** (which is not printed).

GTTIME@

446D will put in the 8-byte buffer (pointed to by 'HL') the time in the ASCII displayable format of 00:00:00. No terminator is written.

GTDATE@

4470 will put in the 8-byte buffer (pointed to by 'HL') the date in the ASCII displayable format of 00/00/00. No terminator is written.

FEXT@

4473 will insert in the filespec (pointed to by 'DE') the 3-byte file extension pointed to by 'HL' (if the filespec does not already contain an extension). The extension pointed to by 'HL' *must* be 3 characters long.

Other DOS Trivia

Have you ever mucked around on a VTOS or TRSDOS system disk with superzap, and you couldn't find the boot messages? The boot messages are in System 0, and are in *negated* form. This means all the characters were negated using the Z-80 NEG operation. The ASCIIZAP program in the back of this book will let you display and modify these. "What's this," you say? "Who says You can't open system files?" Read on.

Tired of those annoying passwords? Here are passwords (I call them override passwords) for the DOSes. These will let you access, kill, modify or whatever to *any* file, even system files, at *any* time. An override password that will work on VTOS 4.0 is B8D. An override password that will work with LDOS (which is VTOS with modifications) is KHE3. An override password that will work with TRSDOS or NEWDOS 2.1 is NV36. NEWDOS/80 does not have an override password, but who cares? It has a system command that lets you disable password checking.

How would you like to disable password checking in VTOS, LDOS or TRSDOS altogether? Well, first VTOS. Write a patch file that's worded as follows, and execute:

PATCH SYS2/SYS.B8D

The zaps here will not disable MASTER password checking when full disk backups are performed. This patch is for VTOS 4.0 only.

Figure 6.15 *VTOS Password Disable Routine*

```
.PATCH TO DISABLE VTOS PASSWORD CHECKING
.18 IS THE HEX NUMBER FOR Z-80 CODE `JR.'
.THIS REPLACES `JR  Z'
.
X'4F1C'=18
.
.END OF PATCH
```

To disable LDOS password checking, write this PATCH file, and execute:

PATCH SYS2/SYS.KHE3.

Figure 6.16 *LDOS Password Disable Routine*

```
.PATCH TO DISABLE LDOS PW CHECKING.
.
X'4F12'=18
.
.
```

To disable TRSDOS password checking, run this Disk BASIC program. This patch will only work with TRSDOS versions 2.2 and 2.3.

Figure 6.17 *TRSDOS Password Disable Routine*

```
10 OPEN"R",1,"SYS2/SYS.NV36": FIELD1,255ASA$
20 GET1,2: B$=A$
30 MID$(B$,100,1)=CHR$(24)
40 LSETA$=B$:PUT1,2
50 END
```

Now you'll never have to mess with those nasty passwords again. I bet it feels better already.

Model III Supplement to Chapter 6
Single-Density Disk Format

This format must be carefully followed in order for a correct format to be accomplished. Any deviation may lead to the last sector getting chopped off or inaccessible sectors, so follow it carefully. Before issuing a format command, you must set up the buffer *exactly* as the track is to be written.

Figure 6.18 Buffer Format

Number of bytes	Hex value	Purpose
36	00	Pre track filler

This block is written *once* per sector. Since we are going to use the 10-sector TRSDOS scheme, this block will be written 10 times. The blocks are written one right after the other.

Figure 6.19 Sector Blocks

1	FE	Sector ID address mark.
1	XX	Track number. This MUST be correct!
1	00	Side number. Use 00.
1	XX	Sector number. This value will be the sector name - i.e., 05 = SECTOR 5.
1	01	Sector length computation value. See text.
1	F7	Generate CRC parity bytes.
17	00	Pre sector data filler.
1	FB	Sector data mark. See text.
256	E5	Initial sector data.
1	F7	Generate sector data CRC.
6	00	Post sector filler.

After all 10 sectors are written, approximately 200 bytes of FF will have to be written, so format your buffer with about 300 bytes of FF, just to be safe.

The sector-name byte is the byte actually used in naming that sector. TRSDOS uses sector names from 0 to 9. You could name the sectors in sequential order 0, 1,

2, 3, 4, 5, 6, 7, 8, 9, but to speed-up sector accessing when doing sequential sector reads, the sector names are interlaced. TRSDOS uses 0, 5, 1, 6, 2, 7, 3, 8, 4, 9. This interlace technique allows more time for the DOS and/or programs to perform calculations before the next sector is accessed. In most cases all 10 sectors can be accessed within 2 revolutions of the disk (where it might otherwise take 3 or 4) because the programs weren't quick enough to 'catch' the next sector as it was coming around.

The 'sector length computation' byte is used by the controller to determine how many bytes the sector data is. Using the IBM (normal TRSDOS) format, sector lengths of 128, 256, 512 and 1024 byte sectors can be achieved by using 0, 1, 2 and 3 respectively for the sector length computation values. There is no way to squeeze ten, 512-byte sectors on a single-density track, so get that out of your mind. Also, if you decide that you may want to use a different sector length, you must adjust the quantity of initial sector data bytes to the length of the sector. You will also have to do some experimenting to see how many sectors of such-and-such length you can fit on a track.

notes

7

TRS-80 Model I Interrupts

What is an interrupt? When you're comfortably sitting in the den, smoking your pipe and watching TV, suddenly there's a knock at the front door, or the phone rings. That's an interrupt. Or your wife hollers. That's a *real* interrupt! (From this point on, wives will be referred to as non-maskable interrupts). But Seriously, the Model I uses a Z-80 as its processor, right? If you're a machine-language programmer, I'm sure you know that by now. The Z-80 has a facility that allows program operation to be interrupted. Pushing the current contents of the Program Counter (PC Reg), the interrupt causes the Z-80 to start executing at a different address until the interrupt is done, or in other words, when a return is executed. It is more or less a 'Forced Call' to an address which we will discuss later.

What could you use an interrupt for? Well, you could have a routine that updates a time clock, provided the interrupts were generated at a constant rate (like the one in the expansion interface). You could also have an interrupt routine that checks for a group of keys to be pushed (like JKL) to perform some function, like a print-the-screen routine.

The Z-80 has 2 types of program interruption. One type has three modes. The first kind of interrupts is called the *maskable* interrupts. Maskable means the interrupts may be enabled and disabled at will by the software by executing DI and EI (disable interrupt and enable interrupt mnemonics, respectively).

Only one mode of the maskable interrupts may be active at any one time. I'm going to explain how interrupt modes 0, 1 and 2 work. I am assuming that interrupts are enabled, and an interrupt is being generated by an external device.

Mode 0 interrupts allow an external device to place a byte on the Z-80 data bus. This byte is usually a RST to one of the restart locations.

Mode 1 interrupts cause the Z-80 to perform a RST to location 38. This is the mode used by the Model I. This will be explained in detail a little further on.

Mode 2 interrupts cause the Z-80 to call a location derived by the lower 8 bits supplied by the device and the upper 8 bits currently in the I Register.

The second type of interrupt is of the *non-maskable* type. This means that if the NMI pin on the Z-80 ever goes to a low voltage level, the Z-80 will force a call to **66**. On the TRS-80 Model I, this is connected to the reset button in the back.

The Z-80 has an input called the Interrupt Request (IR). Every time the Z-80 finishes its current machine cycle (same as one heartbeat to us humans), it logically ANDs the input of the IR with the interrupt enable flip-flop to determine if an interrupt should be serviced. You don't worry about this IR input. This is taken care of by the IR clock inside the expansion interface. This clock generates an IR request once every 25 ms, or 40 times a second.

If the IR is active and the interrupt enable flip-flop is on, the Z-80 executes a **RST 38**, which is the one-byte equivalent of a **CALL 38**. In this system, this address is a ROM address, so in order to control where the interrupt routine is, **38** contains a **JP 4012**, which is a RAM address. This address is the 3-byte vector in which you would place your interrupt routine directive, like **JP 5200** if that's where your routine is.

The interrupt enable flip-flop may be switched on or off at any time by executing the Z-80 opcodes EI and DI; these enable and disable the interrupts, respectively. When this flip-flop is off, the Z-80 will ignore the IR input.

Figure 7.1 *Interrupt Redirection Routine*

```
*****
**      Example of Redirecting the RST 38 Interrupt Vector      **
*****

LD      A,0C3H          ;The JP Instruction
LD      (4012H),A       ;Store JP Opcode
LD      HL,5022H        ;Interrupt Routine Address
LD      (4013H),HL      ;Store Jump Address
```

(new interrupt routine response to interrupt)

There are two memory-mapped addresses associated with reading the interrupt status. This must be done in the interrupt service routine to reset the clock in the expansion interface and reset the FDC interrupt request. The addresses are shown below.

By a coincidence, **37E0** is also the disk select latch, but only when *writing* to that location. When reading, it gives the interrupt status. The graph below shows what the bits are used for.

Figure 7.2 Interrupt Read Addresses

```

7   6   5   4   3   2   1   0
:   :
:   ----- If interrupt was from the FDC.
----- If interrupt was generated from the clock.

```

Bit 6 means the FDC made the request. Usually you ignore the interrupt and return.

Bit 7 means the interrupt was made by the clock.

You should read the FDC stat/cmd 37EC Register on every interrupt to reset it.

In your interrupt routines, you should preserve all registers you will be using in your interrupt routine. Interrupts are automatically disabled after you read 37E0, and must be re-enabled after the routine, if you want the interrupts enabled. Below is a typical interrupt routine.

Figure 7.3 Interrupt Routine

```

*****
**                               Interrupt Routine                               **
*****

      PUSH      AF                ;Save AF.
      LD        A,(37E0H)         ;Clear FCD
      LD        A,(37E0H)         ;Get Int Status
      BIT       6,A               ;FDC Making Int?
      JR        NZ,RETINT         ;Go If So
      BIT       7,A               ;Was It Valid?
      JR        Z,RETINT          ;Go If Not
      LD        A,(3840H)         ;Get KB Byte
      BIT       7,A               ;Break Hit?
      JR        Z,RETINT          ;Go If Not
      JP        440DH             ;JP To DEBUG
RETINT LD        A,(37E0H)         ;Reset Latch
      POP       AF               ;Restore AF
      EI                    ;Enable Int
      RET                      ;Return

```

This is an example of a routine that would jump to debug if the break key were hit.

So, what do you do if you want to link an interrupt routine with the ones already running in the DOSes? Below is a routine that would do just that.

Figure 7.4 Interrupt Routine

```

*****
**                                **
**                                **
*****
**                                **
**                                **
*****

DI                                ;KILL INT
LD      HL,DOSA                   ;GET CURRENT INT ADR
LD      (DOS+1),HL                ;STORE FOR JP
LD      HL,NEWINT                 ;GET NEW ROUTINE
LD      (4013H),HL               ;STORE NEW ADR
EI                                ;ENABLE INT
RET

NEWINT  PUSH    AF                ;SAVE AF
LD      A,(37ECH)                 ;CLEAR FCD
LD      A,(37E0H)                 ;GET INT STATUS
BIT     6,A                       ;FDC MAKING INT?
JR      NZ,RETINT                 ;GO IF SO
BIT     7,A                       ;WAS IT VALID?
JR      Z,RETINT                 ;GO IF NOT

<YOUR ROUTINE(s) GOES HERE>

DOS     POP     AF                ;RESTORE
JP      0                       ;GO TO DOS's
RETINT  LD      A,(37E0H)         ;RESET INT LATCH
EI                                ;ENABLE
RET

```

The DOS address must be right after the instructions that test 37E0 in the DOS's interrupt service routine. You can find DOS interrupt routine through *tracking it down* by looking at the 4012 vector and seeing where it jumps.

You must disassemble routines using RSM, or some other disassembler, to determine where to jump to.

But what about those *NON*-maskable interrupts? The non-maskable interrupts are used in the Model I to reset BASIC or DOS, whichever is appropriate. This is invoked by pressing the reset button on the back of the TRS-80. The non-maskable interrupt cannot be *masked out* or disabled. This feature is of no use to the TRS-80 user other than resetting his computer. When the Z-80 gets an NMI reset, it does a **CALL 66H** instruction, which in this system resets BASIC (or DOS if the expansion interface is connected). The NMI and normal, maskable interrupts have nothing to do with each other.

In the back of this book are some useful interrupt routines.

Model III Supplement To Chapter 7

The Model III Interrupt System

The Model III has a more expanded *maskable* interrupt handling system than the Model I. Maskable interrupts can be generated by the cassette, RS-232, and the real-time clock. Before we go on, below is the list of port assignments for the Model III.

Figure 7.5 Model III Port Assignments

```

***** INTERRUPT PORTS *****

E0 - WRITE - ENABLES VARIOUS DEVICE INTERRUPTS.

7      UNUSED
6      1 ENABLES RS-232 INT ON PARITY/FRAME/OVERRUN ERRORS
5      1 ENABLES RS-232 DATA RECIEVED INT
4      1 ENABLES RS-232 XMIT BUFFER EMPTY
3      1 ENABLE BUS I/O INT (EXT HARD DISK, ETC.)
2      1 ENABLES REAL-TIME CLOCK INT
1      1 ENABLE 1500 BAUD CASSETTE FALLING EDGE INT
0      1 ENABLE 1500 BAUD RISING EDGE INT

E0 - READ - READ INTERRUPT STATUS

7      UNUSED
6      0 = RS-232 PARITY, FRAME, OR OVERRUN ERRORS
5      0 = RS-232 DATA RECEIVED
4      0 = RS-232 XMIT BUFFER IS EMPTY
3      0 = BUS INTERRUPT (EXT HARD DISK, ETC)
2      0 = REAL-TIME CLOCK
1      0 = 1500 BAUD CASSETTE FALLING EDGE
0      0 = 1500 BAUD RISING EDGE INT

EC - READ - RESET REAL-TIME CLOCK

      THIS PORT IS SIMPLY READ BY THE INTERRUPT ROUTINE
      TO RESET THE TIME CLOCK.

E4 - WRITE - SELECT NON-MASKABLE INTERUPT DEVICES

7      1 = ENABLE FDC 'INTRQ' TO NMI
6      1 = ENABLE DRIVE 'TIME-OUT' TO NMI
5-0    UNUSED

E4 - READ - READ NON MASKABLE INTERRUPT STATUS

7      0 = FDC 'INTRQ' OCCURED.
6      0 = DISK 'TIME OUT' OCCURED.
5      0 = RESET BUTTON PRESSED

```

Listing Continued . . .

... Continued Listing

***** DISK CONTROLLER PORTS *****

F0 - READ - FDC STATUS REGISTER
 F0 - WRIT - FDC COMMAND REGISTER
 F1 - READ/WRIT - FDC TRACK REGISTER
 F2 - READ/WRIT - FDC SECTOR REGISTER
 F3 - READ/WRIT - FDC DATA REGISTER

 F4 - WRIT - DISK DRIVE SELECT PORT

 7 1 = DOUBLE-DENSITY, 0=SINGLE
 6 1 = GENERATE WAITS
 5 1 = WRITE PRECOMP
 4 0 = SIDE 0, 1 = SIDE 1 (NOT STANDARD)
 3 SELECT DRIVE 3
 2 SELECT DRIVE 2
 1 SELECT DRIVE 1
 0 SELECT DRIVE 0

***** RS-232 PORTS *****

E8 - READ - MODEM STATUS REGISTER

 7 CLEAR TO SEND, PIN 5 OF DB-25
 6 DATA SET READY, PIN 6
 5 CARRIER DETECT, PIN 8
 4 RING DETECT, PIN 22
 3-2 UNUSED
 1 UART CONTROL REGISTER LOAD STATUS. 1=ON
 0 RECEIVER INPUT, PIN 20

E8 - WRITE - MASTER RESET
 E9 - READ - BAUD RATE SENSE SWITCHES
 E9 - WRITE - SELECT BAUD RATE
 EA - READ - UART STATUS

7 1 = DATA RECEIVED
 6 XMIT HOLDING EMPTY
 5 OVERRUN ERROR
 4 FRAMING ERROR
 3 PARITY ERROR
 2-0 UNUSED

EA - WRITE - UART COMMAND REGISTER

7-6 UNUSED
 5-3 UNUSED
 2 0 = DISABLE XMIT DATA
 1 DATA TERMINAL READY
 0 REQUEST TO SEND

***** LINE PRINTER PORT *****

F8 - READ - PRINTER STATUS

7 BUSY

Listing Continued ...

Continued Listing

```
6      OUT OF PAPER
5      ON LINE
4      FAULT
3-0    UNUSED

F8 - WRITE - WRITE BYTE TO BE PRINTED

***** CASSETTE, AND VIDEO PORTS *****

EC - WRITE - CASSETTE AND VIDEO CONTROL

7-6    UNUSED
5      1 ENABLES VIDEO WAITS
4      1 ENABLES EXTERNAL I/O BUS
3      1 ENABLES ALTERNATE VIDEO CHARACTER SET
2      1 ENABLES 32 CHAR MODE
1      1 TURNS CASSETTE MOTOR ON
0      UNUSED

FF - WRITE - CASSETTE DATA PORT

7-2    UNUSED
1-0    CONTROL CASSETTE OUTPUT LEVEL
```

Handling Model III Interrupts

The maskable interrupt vector is the same for the Model I and Model III. The vector is located at 4012 - 4014. When a maskable interrupt occurs, the Z-80 forces a CALL to location 38. Since this is a ROM address, the ROM then jumps to RAM location 4012. This allows a programmable interrupt handling routine to reside anywhere in memory.

Once your interrupt routine is in control, you can read the INT status from port E0, determine the cause of the INT, and branch to the appropriate subroutine. Keep in mind that if the interrupt was caused by the REAL-TIME CLOCK, you *must* read port EC to reset it. Otherwise, an immediate interrupt will occur once interrupts are enabled again before returning to the interrupted program, thereby causing a system 'HANG'.

To select the desired interrupt options, store the proper bits in port E0.

Figure 7.6 Interrupt Bits in Port E0

```
LD      A,4                ;Enable Real-Time Clock
OUT     (0E0H),A
RET
```

RS-232 Interrupts

If you're familiar with the RS-232, you might want to use the RS-232 interrupt options. There are three different options for the RS-232. These are:

1. INT when PARITY, FRAMING, or OVERRUN error occurs.
2. INT when a character has been recieved.
3. INT when the RS-232 transmitter 'holding tank' is empty.

Port Mapped External Devices and Interrupts

The Model III has hardware installed that permits Z-80 access to external devices via ports **00** through **7F**. Tandy reserves ports **80** through **FF**, and they should not be used for your devices if you want to maintain Tandy compatibility, since Tandy may make available devices that use these addresses. For more information on the hardware aspect, order the Radio Shack Model III service manual, catalog numbers 26-1061/1062/1063.

Once the hardware is connected to the applicable ports, the port bus can be enabled or disabled by switching bit 4 in port **EC**.

There is also an external maskable interrupt tie-in capability, which allows external devices to generate a maskable interrupt, if desired, by using this feature. The external interrupt line may be enabled or disabled by switching bit 3 in port **E0**. Note that even if external interrupts are disabled, you can still read the interrupt status line (bit 4, port **E0**).

Real Time Clock Interrupts

The Model III's real-time clock provides a 25 or 30 Hz signal (for 50 and 60 Hertz AC current respectively) to the maskable interrupt circuits for time clock generation. If interrupts are enabled, and the REAL-TIME CLOCK (RTC) option is set in port **E0**, then a call to **38** is executed. Since this is a ROM address, a jump to RAM address **4012** is performed. This should contain a jump instruction to the software interrupt handler. The interrupt handler can check whether this was an RTC interrupt by testing bit 2 in Port **E0**. If this was, in fact, an RTC interrupt, you would then *reset* the real-time clock by reading Port **EC**. If this is not done, the RTC line will stay on, and when the routine enables the interrupts again, the routine itself will be interrupted. This will either cause a 'hang' in the system or eventual clobbering of the RAM stack, probably within milliseconds.

The Model III already has interrupt handler and real-time clock update/display routines built right in! The interrupt vector routine is located at **3018** (this jumps to **35C2**, where the actual routine is). It branches to different RAM and ROM addresses, depending on what device interrupted it. Below is the table used by Model III interrupt handler.

Figure 7.7 *Model III Interrupt Handler*

Port E0
Bit

0	RISING EDGE 1500 cassette	->	3365H ROM
1	FALLING EDGE 1500 cassette	->	3369H ROM
2	REAL-TIME CLOCK	->	4046H RAM
	Which usually jumps to 3529H. This routine flashes the cursor. It displays the clock in 00:00:00 format if bit 0 at 4210H is set.		
3	EXTERNAL INTERRUPT	->	403DH RAM
4	RS-232 XMIT BUFFER IS EMPTY	->	4206H RAM
5	RS-232 DATA RECEIVED	->	4209H RAM
6	RS-232 RECEIVE ERROR	->	4040H RAM
7	CURRENTLY UNUSED	->	4043H RAM

The 1500-Baud Cassette Interrupts

You probably won't find these too useful, unless you're into cassette operations. When these options are enabled, an interrupt will be generated when the cassette input voltage goes low and/or high (depending on options). You must use your own interrupt vector routine if you desire to use these, because the ROM interrupt vector routine jumps directly to ROM subroutines that control the cassette.

notes

8

Handy Routines, Drivers and Programs

Below are some useful routines and programs I have written. Just key them into EDTASM or BASIC, whichever applies, and use them.

TRSDOS Error Displayer

As previously mentioned in Chapter 6, you may use the TRSDOS error displayer. But if, for some unpredictable reason, you do not want to use TRSDOS's, perhaps because you're writing a new super-doooper disk-zap or something, and you want it to be independent of the DOS, I have included this not-too-sophisticated error displayer. It doesn't do those fancy diagnostics that TRSDOS has. This routine is made to be inherent in a program, and is not used as a DOS overlay.

Figure 8.1 *TRSDOS Error Displayer*

```

00100 ;      TRSDOS COMPATIBLE ERROR MSG DISPLAYER
00110 ;      LD A With Error Code - CALL "ERRMSG"
00111 ;      Uses Regs HL, B, DE (in Call to 33H)
00120 ;
00140 ERRMSG EQU      $      ;ROUTINE START
00170      CP          43      ;OVER MAX ERROR CODE?
00180      JR          C,WW    ;GO IF NOT OVER
00190      LD          A,43     ;LOAD WITH UNDEFINED MSG
00200 WW      LD          B,A    ;PUT ERROR IN B
00210      INC         B        ;BUMP SO 0=1 ETC.
00220      LD          HL,ERRORS ;GET ERROR MSG TABLE
00230 L1      CALL      BYTE    ;SEARCH FOR NEXT MSG
00240      DJNZ        L1       ;DO UNTIL ERROR REACHED
00250      CALL        4467H    ;CALL DOS LINE DISPLAY

```

Listing Continued...

... Continued Listing

00260	LD	A,13	;CARRIAGE RETURN BYTE
00270	JP	33H	;PRINT IT, AND RETURN
00320	BYTE LD	A,(HL)	;GET BYTE
00330	INC	HL	;BUMP
00340	CP	3	;DEL?
00350	RET	Z	;RET IF SO
00360	JR	BYTE	;TRY AGAIN
00370	ERRORS EQU	\$	
00380	DEFB	3	
00390	DEFM	'NO ERROR'	
00400	DEFB	3	
00410	DEFM	'PARITY ERROR DURING ID READ'	
00420	DEFB	3	
00430	DEFM	'SEEK ERROR <READ>'	
00440	DEFB	3	
00450	DEFM	'LOST DATA <READ>'	
00460	DEFB	3	
00470	DEFM	'PARITY ERROR <READ>'	
00480	DEFB	3	
00490	DEFM	'SECTOR NOT FOUND <READ>'	
00500	DEFB	3	
00510	DEFM	'PROTECTED SECTOR'	
00520	DEFB	3	
00530	DEFM	'PROTECTED SECTOR'	
00540	DEFB	3	
00550	DEFM	'DEVICE NOT AVAILABLE'	
00560	DEFB	3	
00570	DEFM	'PARITY ERROR DURING ID WRITE'	
00580	DEFB	3	
00590	DEFM	'SEEK ERROR <WRITE>'	
00600	DEFB	3	
00610	DEFM	'LOST DATA <WRITE>'	
00620	DEFB	3	
00630	DEFM	'PARITY ERROR <WRITE>'	
00640	DEFB	3	
00650	DEFM	'SECTOR NOT FOUND <WRITE>'	
00660	DEFB	3	
00670	DEFM	'DISK DRIVE FAULT <WRITE>'	
00680	DEFB	3	
00690	DEFM	'WRITE PROTECTED DISK'	
00700	DEFB	3	
00710	DEFM	'BAD FILE NUMBER'	
00720	DEFB	3	
00730	DEFM	'DIRECTORY ERROR <READ>'	
00740	DEFB	3	
00750	DEFM	'DIRECTORY ERROR <WRITE>'	
00760	DEFB	3	
00770	DEFM	'ILLEGAL FILESPEC'	
00780	DEFB	3	
00790	DEFM	'GAT READ ERROR'	
00800	DEFB	3	
00810	DEFM	'GET WRITE ERROR'	
00820	DEFB	3	
00830	DEFM	'HIT READ ERROR'	
00840	DEFB	3	
00850	DEFM	'HIT WRITE ERROR'	

Listing Continued...

... Continued Listing

```

00860      DEFB      3
00870      DEFM      'FILE NOT FOUND'
00880      DEFB      3
00890      DEFM      'FILE ACCESS DENIED'
00900      DEFB      3
00910      DEFM      'DIR SPACE FULL'
00920      DEFB      3
00930      DEFM      'DISK SPACE FULL'
00940      DEFB      3
00950      DEFM      'END OF FILE'
00960      DEFB      3
00970      DEFM      'PAST END OF FILE'
00980      DEFB      3
00990      DEFM      'CANT EXTEND FILE'
01000      DEFB      3
01010      DEFM      'PROGRAM NOT FOUND'
01020      DEFB      3
01030      DEFM      'ILLEGAL DRIVE NUMBER'
01040      DEFB      3
01050      DEFM      'DEVICE SPACE FULL'
01060      DEFB      3
01070      DEFM      'NOT AN OBJECT FILE'
01080      DEFB      3
01090      DEFM      'BAD MEMORY'
01100      DEFB      3
01110      DEFM      'TRIED TO LOAD ROM'
01120      DEFB      3
01130      DEFM      'ACCESS TO PROTECTED FILE DENIED'
01140      DEFB      3
01150      DEFM      'FILE NOT OPEN'
01160      DEFB      3
01170      DEFM      'DRIVE NOT READY'
01180      DEFB      3
01190      DEFM      'SYSTEM PROGRAM NOT FOUND'
01200      DEFB      3
01210      DEFM      'PARAMETER ERROR'
01220      DEFB      3
01230      DEFM      'OUT OF MEMORY'
01240      DEFB      3
01250      DEFM      'UNDEFINED ERROR CODE'
01260      DEFB      3
01270 ;
01280 ;      END OF ERROR MESSAGE TABLE

```

Disk Formatter Program

Below is a program that will format a disk. Each track contains 10 standard 256-byte, IBM-compatible sectors named 0 to 9. The sector names are in the SECIND table and may be changed to your whims. Remember, the DISKIO driver in Chapter 5 will read or write *any* sector name and *any* IBM-compatible sector.

The number of tracks is defined in the NUMTK DEFB, and this may be changed. Why not modify this program to ask the *number of tracks* and all of the parameters?

The drive is determined by the DRIVE DEFB, and this must be in the SELECT format (i.e., 1, 2, 4, or 8 for drives 0, 1, 2, or 3).

This formatter does *not* initialize a TRSDOS-compatible directory or BOOT.

Figure 8.2 Disk Format Routine

```

00100 ;      DISK FORMATTER
00120 ;
00130      ORG      5200H
00135 NUMTK  DEFB   40      ;Format 40 tracks
00140 TRACK  NOP          ;Current track counter
00150 DRIVE  DEFB   1      ;Drive select byte for
                        ; drive zero
00160 SECIND DEFB   0      ;Sector names
00170      DEFB   5
00180      DEFB   1
00190      DEFB   6
00200      DEFB   2
00210      DEFB   7
00220      DEFB   3
00230      DEFB   8
00240      DEFB   4
00250      DEFB   9
00260 MESS   DEFM   'HIT ANY KEY TO BEGIN FORMATTING'
00270      DEFB   13
00280 START  EQU    $      ;Start of program
00290      LD      HL,MESS  ;Start of message
00300      CALL    4467H    ;Display line
00310      CALL    49H      ;Wait for key hit
00320      CALL    RESTOR   ;Restore drive's head
00330      LD      HL,USER  ;Get SECTOR data buffer
00340      LD      B,0      ;Load counter 256 bytes
00350 L0     LD      (HL),0E5H ;Write sector bytes
00360      INC     HL        ;Bump ptr
00370      DJNZ    L0        ;Do 256 times.
00380      LD      HL,SEC1   ;Get pre-sector header
00390      LD      B,11      ;# bytes to FF
00400 L02    LD      (HL),0FFH ;FF bytes
00410      INC     HL        ;Bump ptr
00420      DJNZ    L02        ;Do 11 times.
00430      LD      HL,DATA   ;Get start of format
                        ; data
00440      LD      (HL),0FCH  ;Write index mark
00450      INC     HL        ;Bump ptr
00460      LD      B,10      ;# of track header bytes
00470 L1     LD      (HL),0FFH ;Init to FF
00480      INC     HL        ;Bump ptr
00490      DJNZ    L1        ;Do 10 times.
00500      LD      B,6        ;# of zeroes to init

```

Listing Continued...

Listing Continued...

```

00510 L11    LD      (HL),0      ;Zero bytes
00520        INC      HL        ;Bump ptr
00530        DJNZ     L11       ;Do 6 times
00540        LD      B,4        ;# of header bytes.
00550 L2     LD      (HL),0FFH   ;Init to FF
00560        INC      HL        ;Bump ptr
00570        DJNZ     L2        ;Do 4 times
00580        EX      DE,HL      ;Get data start in DE
00590 ;
00600        LD      A,10       ;# sectors to set-up
00610 L3     LD      HL,SECTOR   ;Get sector ID overhead
00620        LD      BC,287     ;# bytes to transfer
00630        LDIR          ;Move into buffer
00640        DEC      A         ;Dec # sectors counter
00650        JR      NZ,L3      ;Go move another sector
                                ; if counter not 0
00660        LD      B,0       ;256 trailing track
                                ; bytes.
00670        EX      DE,HL      ;Get end in HL

```

These bytes are the trailing bytes after the last sector. They are written until the FDC is done.

```

00680 L5     LD      (HL),0FFH   ;Init to FF
00690        INC      HL        ;Bump ptr
00700        DJNZ     L5        ;Do 256 times
00710        JR      CTRL      ;Go start format
00720 ;
00730 MTK    LD      HL,DATA     ;Get start of data
                                ; buffer
00740        LD      DE,21      ;# of track overhead
                                ; bytes.
00750        ADD     HL,DE      ;Pass up track header
00760        LD      DE,7       ;Amt to bump to posn
                                ; track
00770        ADD     HL,DE      ;Position over track
00780        LD      DE,SECIND   ;Get sector names buff
00790        LD      B,10       ;# of sectors to name
00800 L4     LD      A,(TRACK)   ;Get current track #
00810        LD      (HL),A     ;Put track in sector
                                ; ID.
00820        INC     HL         ;Position over sector
00830        INC     HL
00840        LD      A,(DE)     ;Get sector name
00850        LD      (HL),A     ;Put in sector ID
00860        INC     DE        ;Bump to next sec name
00870        PUSH    DE        ;Save name pointer
00880        LD      DE,285     ;Amt to posn over next
                                ; sector ID
00890        ADD     HL,DE      ;Position
00900        POP     DE         ;Get names again
00910        DJNZ     L4        ;Loop 10 sec not done
00920        RET              ;Ret to CTRL
00930 ;
00940 ;      CONTROLLER
00950 ;

```

... Continued Listing

```

00960 CTRL EQU $
00970 EQQ CALL MTK ;Init sector names and
                                ; track in sectors ID's
00980 CALL EXEC ;Go format current trk.
00990 LD A,(TRACK) ;Get current track.
01000 INC A ;Add 1
01010 LD (TRACK);A ;Store in counter
01015 LD C,A ;Put in C
01017 LD A,(NUMTK) ;Get max number of TKS
01020 SUB C ;Max yet?
01030 JR Z,EXQ ;Go if max reached
01040 CALL STEPIN ;Step head in once.
01050 JR EQQ ;Start process for next
                                ; track
01060 EXQ CALL RESTOR ;Restore head
01070 XOR A ;Clear for no error ret
                                ; to DOS
01080 EI ;Enable interrupts
01090 RET ;Ret to DOS
01100 ;
01110 EXEC EQU $
01120 LD DE,DATA ;Get track set-up
01130 LD HL,37ECH ;FDC cmd/stat address
01140 CALL SELECT ;Select drive
01150 LD A,(DE) ;Get first data byte
01160 INC DE ;Bump data ptr
01170 DI ;Disable interrupts
01180 LD (HL),0F4H ;Issue write track cmd
01190 CALL WAIT ;Wait for FDC to act
01200 ;
01210 LOOP EQU $
01220 BIT 1,(HL) ;Does FDC request a
                                ; byte?
01230 JR NZ,PUT ;Go if so
01240 LOOP1 BIT 0,(HL) ;Format done?
01250 RET Z ;Ret if so
01260 JR LOOP ;Loop if not
01270 PUT LD (37EFH),A ;Transfer byte to FDC
01280 INC DE ;Bump data ptr
01290 LD A,(DE) ;Get next data byte
01300 JR LOOP1 ;Continue

```

Start of sector data set-up

```

01320 SECTOR NOP
01330 NOP
01340 NOP
01350 NOP
01360 NOP
01370 NOP
01380 DEFB 0FEH ;ID address mark
01390 NOP ;Spot for track
01400 DEFB 1 ;Side number
01410 NOP ;Spot for sector
01420 DEFB 1 ;IBM sector compute byte

```

Listing Continued...

... Continued Listing

01430		DEFB	0F7H	; for 256 byte sector
				;Gen CRC for address
				; byte
				;For filler
01440	SECL	DEFS	11	
01450		NOP		
01460		NOP		
01470		NOP		
01480		NOP		
01490		NOP		
01500		NOP		
01510		DEFB	0FBH	;Data address mark
01520	USER	DEFS	256	;For E5 user data
01530		DEFB	0F7H	;Gen CRC for data
01540	RESTOR	CALL	SELECT	;Select drive
01550		LD	A,0D0H	;Reset FDC cmd
01560		LD	(37ECH),A	;Reset FDC
01570		CALL	WAIT	;Give FDC a chance
01580		LD	A,3	;Restor cmd, 40 ms.
01590		LD	(37ECH),A	;Issue restore cmd
01600		CALL	WAIT	;Let FDC respond
01610	RES1	LD	A,(37ECH)	;Get status
01620		RRCA		;Shift busy into C
01630		RET	NC	;Ret if restor done
01640		CALL	SELECT	;Select drive
01650		JR	RES1	;Loop
01660	SELECT	LD	A,(37ECH)	;Get status
01670		PUSH	AF	;Save
01680		LD	A,(DRIVE)	;Get select code
01690		LD	(37E0H),A	;Select drive
01700		POP	AF	;Get old status
01710		RLCA		;Shift not ready bit
				; into C flag
01720		RET	NC	;Ret if drives were
				; already rotating
01730		PUSH	BC	;Save reg
01740		LD	BC,0A000H	;Delay for 1 sec to let
				; drive motors come up
				; to speed.
01750		CALL	60H	;Call ROM BC delay
				; routine
01760		POP	BC	;Restore BC
01770		RET		;Return
01780	;			
01790	STEPIN	CALL	SELECT	;Select drive
01800		LD	A,0D0H	;Reset FDC cmd
01810		LD	(37ECH),A	;Reset FDC
01830		LD	A,43H	;Step-in cmd, no upd
01840				; no verf, 40 ms rate.
01850		LD	(37ECH),A	;Issue step-in cmd
01860		CALL	WAIT	;Get FDC act
01870	BUSY	LD	A,(37ECH)	;Get status
01880		RRCA		;Shift busy into carry
01890		RET	NC	;Ret if not busy
01900		JR	BUSY	;Loop until not busy
01910	WAIT	PUSH	HL	;Waste a few microsecs
01920		POP	HL	; to let FDC react to

Listing Continued...

... Continued Listing

```

01930      PUSH    HL          ; cmd
01940      POP     HL
01950      RET
01970 DATA EQU      $          ; End of program
                                   ; track data assembled
                                   ; here

01980      END      'START

```

PASSFIND, Password Finder

Below is a small program that will take a password encode and display passwords that will work for that encode. Encodes must be entered in MSB, LSB order. For example, if the encode looks like 9642 in the file's directory spot, that is really the LSB, MSB form of 4296, and it must be entered 4296 in this program.

The routine tests about 1300 passwords per second and usually takes about 1 minute before a match will be found. After the match is printed, the routine will continue printing out all possible matches until you hit the reset button. This is the program I used to decode the VTOS and TRSDOS override passwords.

Figure 8.3 Password Display Routine

```

00100 ;      PASSFIND - CALCULATES THE PASSWORD OF A FILE.
00120 ;
00130      ORG      5200H
00140 ENCODE DEFW  0          ; Encode storage
00150 INITM  DEFM  'PASSWORD FINDER PROGRAM'
00160      DEFB    10
00170      DEFB    13
00180 INBUF  DEFS    5          ; Line input buffer
00190 PBUFF  DEFS    8          ; Password assembly
                                   ; Print buffer
00200      DEFB    3          ; Print terminator
00210 XFER   LD     HL,INITM    ; Get program announce
00220      CALL   4467H         ; Display
00230 MAIN   LD     A,'*'      ; Get prompt char
00240      CALL   33H          ; Display
00260      LD     HL,INBUF      ; Get input buffer
00270      LD     B,4          ; Max # of chars to input
00280      CALL   40H          ; Go line input
00290      JP     C,402DH       ; Go if break hit
00300      LD     A,B          ; Get # chars input
00310      OR     A            ; Set flags
00320      JR     Z,MAIN        ; Go back if enter only hit
00330      CALL   HEXIN        ; Convert input to hex
00340      JR     C,MAIN        ; Go if any chars were
                                   ; not hex digits
00360      LD     (ENCODE),DE   ; Store encode
00370      LD     HL,PBUFF      ; Get PW assembly buffer

```

Listing Continued...

... Continued Listing

```

00380      LD      (HL), 'A'      ;Init with first PW
00390      LD      B,7            ;# of chars left
00400      INC     HL             ;Bump assembly ptr
00410  I1      LD      (HL), ' '   ;Put in spaces
00420      INC     HL             ;Bump ptr
00430      DJNZ    I1            ;Do 7 times
00440      JP      EXEC          ;Go start compares
00450  CONV    EQU      $
00460      CP      '0'           ;Is it num?
00470      JR      C,C1          ;Go if under '0'
00480      CP      'G'           ;Over hex?
00490      JR      NC,C1         ;Go if over
00500      CP      ':'           ;0-9?
00510      JR      C,C2          ;Go if so
00520      CP      'A'           ;A-F?
00530      JR      NC,C3         ;Go if so
00540  C1      SCF              ;Set error flag
00550      RET                  ;Ret
00580  C2      SUB      30H       ;Adjust 'dec' to real
00590      OR       A            ;Kill C flag
00600      RET                  ;Ret
00610  C3      SUB      37H       ;Adjust hex digit
00620      OR       A            ;Kill C flag
00630      RET                  ;Ret
00640  HEXIN   EQU      $
00650      LD      A,(HL)        ;Put hex input in DE
00660      INC     HL            ;Get next byte
00670      CP      13           ;Bump input ptr
00680      RET      Z            ;Terminate?
00690      CALL    CONV          ;Ret if end of input
00700      RET      C            ;Convert byte to real
00710      PUSH    HL           ;Ret if invalid chars
00720      EX      DE,HL         ;Save input ptr
00730      ADD     HL,HL         ;Get current accum in HL
00740      ADD     HL,HL         ;Mult accum by 16
00750      ADD     HL,HL         ;Shift HL left 4 times
00760      ADD     HL,HL
00770      LD      D,0           ;Zero
00780      LD      E,A           ;Get last digit
00790      ADD     HL,DE         ;Add to accum
00800      EX      DE,HL         ;Put in DE
00810      POP     HL           ;Get input ptr
00820      JR      HEXIN        ;Process next byte
00830 ;
00840 ;      EXECUTES PW CALC
00850 ;
00860  EXEC    EQU      $
00870      CALL    HASH          ;Compute encode from
                                ; current PW in buffer
00880      LD      DE,(ENCODE)   ;Get encode to compare
                                ; against
00890      RST      18H          ;Compare using ROM
                                ; compare DE to HL sub.
00900      CALL    Z,MATCH       ;Go display if match
00910      CALL    UPDATE        ;Go bump current PW
00920      JP      C,GOGO        ;Go if end of test

```

Listing Continued...

... Continued Listing

```

00930          JR      EXEC
00935 ;
00940 ;CREATES A TWO BYTE HASH CODE IN HL
00970 ;
00980 ;
00990 ;      ON ENTRY : DE -> TEXT , B = # OF BYTES
01000 ;
01010 HASH      LD      B,8          ;8 CHARS
01020          LD      DE,PBUFF+7    ;Point to end of PW buff
01030          LD      HL,0FFFFH     ;Init HL
01040 HCl      LD      A,(DE)        ;Get byte
01050          PUSH    DE            ;Save PW ptr
01060          LD      D,A            ;Get char in D
01070          LD      E,H            ;Get cur H in E
01080          LD      A,L            ;Put L in A
01090          AND     7              ;Keep low 3 bits
01100          RRCA
01110          RRCA
01120          RRCA
01130          XOR     L              ;XOR with L
01140          LD      L,A            ;Put in L
01150          LD      H,0            ;Zero H
01160          ADD     HL,HL           ;Shift HL left 4 times
01170          ADD     HL,HL
01180          ADD     HL,HL
01190          ADD     HL,HL
01200          XOR     H              ;Xor A with H
01210          XOR     D              ;Xor A with D char in
01220          LD      D,A            ;Put in D result high byte
01230          LD      A,L            ;Put L in A
01240          ADD     HL,HL           ;Shift HL left once
01250          XOR     H              ;Xor A with H
01260          XOR     E              ;Xor with E
01270          LD      E,A            ;Put in E result low byte
01280          EX     DE,HL           ;Put encode in HL
01290          POP     DE             ;Get ptr
01300          DEC     DE             ;Dec ptr
01310          DJNZ   HCl            ;Do 8 times
01320          RET                  ;Ret with encode in HL
01330 UPDATE   EQU     $
01340          LD      B,8            ;Number of char in PW
01350          LD      HL,PBUFF        ;Get PW buff
01360          XOR     A              ;Clear
01370          LD      (FLAG),A        ;Flag=0 on first char
                                      ;Because char must be
                                      ;A-Z as defined by DOS
01380 U0       LD      A,(HL)        ;Get next char
01390          CALL    BUMP           ;Advance one
01400          RET     NC             ;Ret if done - else must
                                      ;bump next byte also
01410          INC     HL             ;Bump ptr
01420          DJNZ   U0             ;Do max of 8 times
01430          SCF                    ;Test done

```

Listing Continued...

... Continued Listing

```

01440      RET                      ;Ret
01450 BUMP  CP      '9'            ;Time to carry over?
01460      JR      Z,U1           ;Go if so
01470      CP      'Z'            ;Make number yet?
01480      JR      Z,U2           ;Go if so
01490      CP      ','            ;Is it a space?
01500      JR      Z,U3           ;Go if so
01510      INC     A              ;Bump char
01520      LD      (HL),A         ;Put in PW buffer
01530      RET                      ;Ret
01540 U1    LD      A,'A'         ;A ascii
01550      LD      (HL),A         ;Put in PW buff
01570      RET                      ;Ret
01580 U2    LD      A,(FLAG)      ;Is this first char?
01590      OR      A              ;Set flag
01600      JR      NZ,U8          ;Go if not
01610      INC     A              ;Make A=1
01620      LD      (FLAG),A       ;Put in flag
01630      LD      A,'A'         ;Make char A
01640      JR      U80            ;Cont
01650 U8    LD      A,'0'         ;Load with 0 ascii
01660 U80   LD      (HL),A       ;Store char
01670      SCF                    ;Set overflow flag
01680      RET                      ;Ret
01690 U3    LD      A,'0'         ;Load 0 ascii
01700      LD      (HL),A       ;Store char
01710      OR      A              ;Set flags
01720      RET                      ;Ret
01730 FLAG  NOP                    ;Flag byte
01740 GOGO  JP      402DH         ;Jump to DOS

```

Prints Out Match

```

01750 MATCH LD      HL,PBUFF      ;Get PW to print
01760      CALL    4467H          ;Display
01770      LD      A,', '         ;Get comma ascii
01780      JP      33H            ;Display and ret
01790      END      XFER          ;Start of program

```

LOADER/BAS — Object File Load Format Displayer

This is a DISK BASIC program that will display the load format of a machine-language file. It will run under any of the DISK BASICs for the Model I. I have not tested it, but it will probably run under the Model III also.

When you first run the program, the prompt will come like this:

Figure 8.4 Program Prompt

FILE LOAD FORMAT DISPLAYER V2.1

FILESPEC >

Answer the prompt with the name of the object file whose load format you wish to display. Try entering SYS1/SYS. Now the disks will fire, lightning will strike, and the display will end up looking like Figure 8.5 below. (This is NEWDOS 80 SYS1/SYS. Yours may be different.)

Figure 8.5 *SYS1/SYS Display*

```

FILE LOAD FORMAT DISPLAYER V2.1 - CURRENT FILE : SYS1/SYS

254 BYTES LOADED A 4D00 / 19712 - SECTOR 0 , BYTE 04
254 BYTES LOADED A 4DFE / 19966 - SECTOR 1 , BYTE 06
254 BYTES LOADED A 4EFC / 20220 - SECTOR 2 , BYTE 08
254 BYTES LOADED A 4FFA / 20474 - SECTOR 3 , BYTE 0A
254 BYTES LOADED A 50F8 / 20728 - SECTOR 4 , BYTE 0C

HIT <ENTER> TO CONTINUE

```

The SECTOR, BYTE number is the relative sector and byte in which the first byte of this object block (not the loader codes) is stored upon the disk. This is useful for tracking down bytes to modify after disassembling the program.

After hitting enter, the screen will clear and come up looking like the figure below.

Figure 8.6 *DOS Transfer Address Display*

```

FILE LOAD FORMAT DISPLAYER V2.1 - CURRENT FILE : SYS1/SYS

TRANSFER ADDRESS = 4D00 / 19712
TOTAL NUMBER OF BYTES LOADED = 04E0 / 1248

HIT <ENTER> TO CONTINUE

```

This displays the DOS transfer address and the number of object bytes that are loaded. This does not include any header bytes. If you encounter any HEADER bytes, the header block will be described as follows.

8 HEADER BYTES

When keying-in this program, it is unnecessary to put in a line feed after every colon. I did this for readability's sake.

If you just hit enter when the FILESPEC prompt is displayed, it will jump to DOS READY via CMD "S". Also, if you press the up arrow key while LOADER is printing out, the current print out will be terminated, and come back up with the FILESPEC prompt.

Figure 8.5 *LOADER/BAS Routine*

```

10 '          LOADER - VERSION 2.1
20 '          DISPLAYS LOAD FORMAT FOR AN OBJECT FILE
30 '
35 CLS

40 CLEAR2000:
   U$="###":
   O$="#####":
   PRINT@0,"FILE LOAD FORMAT DISPLAYER V2.1"

50 PRINT@128,CHR$(31);:
   LINEINPUT"FILESPEC >";F$:
   IFF$=""THENCMD"S

60 ONERRORGOTO70:
   OPEN"I",1,F$:
   CLOSE:
   OPEN"R",1,F$:
   FIELD1,255ASA$:
   GOTO80

70 CMD"E":
   FORQ=1TO500:
   NEXT:
   RESUME50

80 PRINT@32,"- CURRENT FILE : "F$;:
   PRINT@128,"":
   ONERRORGOTO0:
   X=PEEK(VARPTR(A$)+1)+(PEEK(VARPTR(A$)+2)*256):
   Y=X

90 GET1,1:
   S=2:
   PRINT@128.,CHR$(31);
   ' GET FIRST SECTOR

100 '

110 IFPEEK(&H3840)AND8THENCLOSE:
   PRINT@34,"ABORT DURING : "F$CHR$(30);:
   RUN40
   ELSEGOSUB4000:
   GOSUB 200:
   IF A<>1 THEN 150

120 GOSUB 200:
   C=A:
   IF C=0 THEN C=256
   ELSE IF C<3 THEN C=260-C

```

... Continued Listing

... Continued Listing

```

130 C=C-2:
    CB=CB+C:
    GOSUB 200:
    AD=A:
    GOSUB 200:
    A=A*256 :
    AD=AD+A

140 PRINTUSINGU$;C;:
    PRINT" BYTES LOADED AT ";:
    GOSUB 220:
    PRINT" /";:
    PRINTAD" - SECTOR"S-2", BYTE ";:
    A=B:
    GOSUB250:
    PRINT:
    GOSUB280:
    GOTO110

150 IF A<>2 THEN 180 ' GO IF NOT XFER ADDRESS

160 GOSUB200:
    GOSUB200:
    AD=A:
    GOSUB200:
    A=A*256:
    AD=AD+A

170 CLOSE:
    GOSUB4006:
    PRINT"TRANSFER ADDRESS = ";:
    GOSUB 220:
    PRINT" /";:
    PRINTAD:
    PRINT"TOTAL NUMBER OF BYTES LOADED = ";:
    AD=CB:
    GOSUB220:
    PRINT" /";:
    PRINTCB:
    PRINT:
    GOSUB4006:
    PRINT@34,"LAST FILE : "F$CHR$(30);:
    RUN40

180 GOSUB200:
    C=A:
    IFC=0 THEN C=254

190 PRINTUSINGU$;C;:
    PRINT" HEADER BYTES":
    GOSUB280:
    GOTO110

200 A=PEEK(X+B):
    B=B+1:
    
```

Listing Continued . .

... Continued Listing

```

        IFB=256 THEN 210
        ELSE RETURN

210  B=0:
      GET1,S:
      S=S+1:
      RETURN

220  'PRINTS AD IN HEX

221  IF AD>32767 THEN AD=AD-65536

230  L=AD AND 255:
      M=INT((AD/256) AND 255)

240  A=M:
      GOSUB 250:
      A=L:
      GOSUB 250:
      RETURN

250  Z=(A/16) AND 15:
      GOSUB 260:
      Z=A AND 15:
      GOSUB 260:
      RETURN

260  IF Z>9 THEN 270
      ELSE PRINT CHR$(Z+48);:
      RETURN

270  PRINT CHR$(Z+55);:
      RETURN

280  IF B+C>255 THEN 290
      ELSE B=B+C:
      RETURN

290  B=B+C:
      B=B-256:
      GET1,S:
      S=S+1:
      RETURN

4000  IF PEEK(&H4020)>&H40 AND PEEK(&H4021)>&H3E THEN 4006
      ELSE RETURN

4006  PRINT:PRINT "HIT <ENTER> TO CONTINUE";

4030  A$=INKEY$:
      IF A$=CHR$(13) THEN 4040
      ELSE 4030

4040  PRINT@128,CHR$(31);:
      RETURN

```

ASCIIIZAP — Modify File's Sector in ASCII

This machine-language program will display a file's sectors in hex and ASCII format, and allow modification of the file in the ASCII mode. There is definitely room for more functions in this program. Try your hand at *sprucing it up* a bit.

After running the program, answer the filespec prompt. It will come up and ask what sector to display. After answering that, it will read the sector, and display it in hex and ASCII. Now you are in the COMMAND MODE. This is denoted by the asterisk in the upper-left corner of the screen.

Using the “,” and the “-” keys, you may scroll sectors of the file; “,” is forward, and “-” is backward.

When you wish to modify a sector type in Mnn while in the command mode (nn is the 2 hex digits of the position you want to start modifying). For example, M42 would start the cursor blinking in byte position 42. The cursor will blink over the character you wish to modify. While in the modify mode, the only four characters that will not be actually written are: left arrow, right arrow, break, and enter. Left and right arrow move the cursor left and right. The break key will terminate the modifications, re-read the sector, and put you back in the command mode. Enter will clear the screen and ask you if the modified sector should be written yet. If you answer yes, it will write the modified sector and re-display the sector, putting you back in the command mode. If you answer no, it will just re-display the buffer, with all modifications, and put you back into the command mode.

If you hit break while in the command mode, the relative sector prompt will come up again. If you hit break at this point, the filespec prompt will come up. If you hit break now, the program will jump back to DOS READY via 402D.

While in the command mode, if N is pressed, the contents of the buffer will be NEGATED using the Z-80 NEG operation. This is useful for detecting negated program header messages like VEE-TOS or TRISS-DOS system number zero. It is also useful for hiding you own scrambled messages. The negate function may be substituted for your own brand of scrambling.

Figure 8.6 ASCIIIZAP Routine

```

00100 ;          ASCIIIZAP - MODIFIES A FILE IN HEX/ASCII
00110 ;
00120 ;
00130          ORG          5200H
00135 ;
00140 BUFFER EQU          $          ; I/O BUFFER
00150          CALL        1C9H          ; CLS
00160          LD          HL, (400DH)    ; GET BREAK
00170          LD          (BTEST), HL    ; STORE IT
00180          LD          HL, BREAKT     ; NEW ROUTINE
00190          LD          (400DH), HL    ; STORE
00200          LD          HL, INITM      ; GET INIT MESSAGE
00210          CALL        4467H          ; DISPLAY
00220          JP          ASK            ; GO TO ROUTINE
00230 INITM   DEFM        'ASCIIIZAP VERSION 1.3 - FILE MODIFY PROGRAM'
```

Listing Continued . . .

... Continued Listing

```

00240      DEFB      10
00250      DEFM      '(C/P) 1981 BY WAGSOFT - ALL RIGHTS RESERVED'
00260      DEFB      10
00270      DEFB      13
00280
00290      ORG       5300H      ;PROGRAM START
00300 ASKM      DEFM      'FILESPEC? '
00310      DEFB      3
00320 MES1      DEFM      'RELATIVE SECTOR? '
00330      DEFB      3
00340 DCB       EQU       0F100H      ;FILE DCB
00350 INBUFF    DEFS      10          ;INPUT BUFFER
00360 SEC       DEFW      0           ;CURRENT SECTOR
00370 POSN      NOP
00380 DISKER    EQU       $           ;HANDLES DISK ERROR
00390      OR        080H            ;FOR RET,NO DIAG
00400      JP        4409H          ;DIS & RET
00410 BREAKT    CP        1           ;BREAK?
00420      RET       Z              ;RET IF SO
00430      DEFB      0C3H          ;JP OPCODE
00440 BTEST     DEFW      0           ;OVERLAY ADR
00450 BOUT      LD        HL,(BTEST) ;GET LOADER
00460      LD        (400DH),HL      ;RESTOR
00470      JP        402DH          ;JP TO DOS
00480 DISKAB    CALL      DISKER      ;DISPLAY
00490 MAIN      EQU       $
00500      LD        SP,41FCH        ;FIX STACK
00510      CALL      1C9H           ;CLS
00520 ASK       LD        HL,ASKM     ;GET MESSAGE
00530      CALL      4467H          ;DISPLAY
00540      LD        HL,DCB          ;FILE DCB
00550      LD        B,22           ;MAX INPUT
00560      CALL      40H            ;LINE INPUT
00570      JP        C,BOUT         ;ABORT
00580      LD        A,B            ;GET AMT
00590      OR        A              ;SET Z
00600      JR        Z,ASK          ;GO IF ENTER HIT
00610      EX        DE,HL          ;PUT IN DE
00620      LD        B,0            ;LRL=256
00630      LD        HL,BUFFER      ;I/O BUFFER
00640      CALL      4424H          ;OPEN FILE
00650      JR        Z,JXP          ;GO IF OK
00660      CALL      DISKER          ;HANDLE ERROR
00670      JR        ASK            ;TRY AGAIN
00680 JXP       INC        DE        ;BUMP TO BYTE 2
00690      LD        A,(DE)         ;GET IT
00700      SET       6,A            ;SET IT
00710      LD        (DE),A         ;DO IT
00720      JR        J1X           ;CONT
00730 ASKS      EQU       $
00740 J1        CALL      1C9H      ;CLS
00750 J1X       LD        HL,MES1     ;SECTOR
00760      CALL      4467H          ;DIS
00770      LD        HL,INBUFF       ;INPUT
00780      LD        B,3             ;MAX IN

```

Listing Continued...

... Continued Listing

```

00790      CALL      40H      ;LINEIN
00800      JP        C,MAIN   ;GO IF FIN
00810      CALL      GETINP   ;GET NUMBER
00820      PUSH      DE      ;PASS TO BC
00830      POP       BC      ;GET REC
00840      LD        (SEC),BC ;SAVE
00850  REREAD  LD        BC,(SEC) ;GET CURRENT
00860      LD        DE,DCB   ;DE=DCB
00870      CALL      4442H   ;POSN TO REC
00880      JR        Z,J2    ;GO IF OK
00890      CALL      ERROR   ;GOTO ERROR
00900      JR        J1      ;ASK NEXT SEC
00910  J2      CALL      4436H ;READ
00920      JR        Z,JX    ;GO IF OK
00930      CP        6       ;PROT SEC?
00940      JR        Z,JX    ;OK
00950      CALL      ERROR   ;DIS
00960      JR        J1      ;DISPLAY BUFFER
00970  DISBUF  EQU       $
00980  JX      CALL      1C9H ;CLS
00990  JX1     CALL      PAGE ;DISPLAY
01000      XOR       A       ;CLR
01010      LD        (POSN),A ;STORE POSN
01020      JP        KEYIN   ;GO TEST
01025 ;
01030 ;
01040 ;
01050 ;
01060 ;
01070 ;
01080  HEXBYT  EQU       $
01090      PUSH      AF      ;SAVE BYTE
01100      RLCA          ;SHIFT MSN TO LSN
01110      RLCA
01120      RLCA
01130      RLCA
01140      CALL      PUTNIB   ;CONVERT NIBBLE
01150      POP       AF      ;RESTORE BYTE
01160      PUSH      AF      ;RE-SAVE
01170      CALL      PUTNIB   ;CONVERT NIBBLE
01180      POP       AF      ;RESTORE BYTE
01190      RET            ;RETURN
01200  PUTNIB  EQU       $
01210      AND      0FH      ;STRIP NIBBLE
01220      CP        10      ;OVER DECIMAL?
01230      JR        NC,PNAAL ;GO IF SO
01240      ADD      A,30H     ;ADD TO ASCII DISPLAY
01250  PNAAL   LD        (HL),A ;STORE BYTE
01260      INC      HL       ;BUMP PTR
01270      RET            ;RETURN
01280  PNAAL   ADD      A,55   ;FOR HEX AJUST
01290      JR        PNAAL2   ;PUT & RET
01295 ;
01300 ;
01310 ;
ROUTINE : NEXTC
GET NEXT NON-SPACE CHAR FROM (HL) STREAM

```

Listing Continued...

... Continued Listing

```

01320 ;      EXIT HL IS POSN OVER CHAR. C SET IF CHAR WAS CR
01330 ;
01340 NEXTC  LD      A,(HL)      ;GET CURRENT CHAR
01350      INC      HL          ;BUMP TO NEXT CHAR
01360      CP      ' '          ;SPACE?
01370      JR      NZ,NEXTC1     ;GO IF NOT
01380      JR      NEXTC         ;GO IF SPACE
01390 NEXTC1 CP      13          ;C/R?
01400      JR      Z,NEXTC2     ;GO IF C/R
01410      OR      A            ;RESET CARRY
01420      RET                     ;RETURN
01430 NEXTC2 SCF                ;SET CARRY
01440      RET                     ;RETURN
01445 ;
01450 ;      TAKES NEXT DECIMAL INPUT AT (HL) AND PUTS IT IN
01460 ;      DE. C SET IS NUMBER WAS BIGGER THAN 65530
01470 ;      ALL REGS USE. HL=CHAR AFTER LAST NUMBER
01480 ;
01490 GETINP EQU      $
01500      LD      DE,0          ;ZERO
01510 GETIN1 CALL     NEXTC     ;GET NEXT CHAR
01520      CP      13          ;C/R?
01530      RET      Z          ;RETURN IF EOL
01540      CP      '0'         ;UNDER 0?
01550      JR      C,GETEND     ;RETURN IF END OF NUMBER
01560      CP      ':'         ;OVER 9?
01570      RET      NC         ;RETURN IF OVER 9
01580      PUSH    HL          ;SAVE LINE PTR
01590      LD      HL,1998H     ;GET # TO TEST
01600      EX      AF,AF'       ;SAVE CHAR
01610      RST      18H         ;COMPARE
01620      JR      C,GETBIG     ;GO IF TOO BIG
01630      EX      AF,AF'       ;RESTORE #
01640      EX      DE,HL        ;SWITCH FOR ADD
01650      PUSH    HL          ;PASS TO DE
01660      POP      DE          ;GET CUR #
01670      ADD     HL,HL        ;TIMES BY 10
01680      ADD     HL,HL
01690      ADD     HL,DE
01700      ADD     HL,HL
01710      SUB     30H          ;ADJUST TO NUMBER
01720      LD      E,A          ;PUT IN DE
01730      LD      D,0          ;CLEAR
01740      ADD     HL,DE         ;ADD TO ACCUM
01750      EX      DE,HL        ;PUT IN DE
01760      POP      HL          ;RESTORE PTR
01770      JR      GETIN1      ;LOOP BACK
01780 GETBIG POP      HL        ;RESTORE PTR
01790      EX      AF,AF'       ;GET THIS CHAR
01800      SCF                ;TOO BIG ERROR
01810      RET                     ;RETURN
01820 GETEND OR      A          ;CLEAR CARRY
01830      RET                     ;RETURN
01840 ;
01850 ;      DISPLAY BUFFER TO SCREEN

```

Listing Continued ...

... Continued Listing

```

01860 ;
01870 PAGE EQU $
01880 LD HL,3C08H ;START OF DISPLAY
01890 LD DE,BUFFER ;I/O BUF
01900 LD B,16 ;LINE TO DISPLAY
01910 J3 PUSH BC ;SAVE COUNT
01920 PUSH DE ;SAVE BUFFER
01930 CALL LINE ;LINE HEX
01940 POP DE ;RESTOR
01950 CALL LINEA ;LINE ASCII
01960 LD BC,8 ;AMT TO ADD
01970 ADD HL,BC ;PUT TO LINE
01980 POP BC ;GET COUNT
01990 DJNZ J3 ;LOOP 16 TIMES
02000 LD HL,3C04H ;FOR BYTE ADR
02010 LD B,16 ;DO 16 TIMES
02020 XOR A ;CLR
02030 LEW CALL HEXBYT ;DISPLAY
02040 ADD A,10H ;BUMP TO NEXT LINE
02050 LD DE,62 ;BUMP ONE LINE
02060 ADD HL,DE ;ADD ONE LINE
02070 DJNZ LEW ;LOOP
02080 RET ;RETURN
02090 LINE LD B,8 ;8 PAIRS
02100 LW1 LD A,(DE) ;GET BYTE
02110 CALL HEXBYT ;DISPAL
02120 INC DE ;BUMP
02130 LD A,(DE) ;GET NEXT
02140 INC DE ;BUMP BUFF
02150 CALL HEXBYT ;DISPLAY
02160 INC HL ;BUMP
02170 DJNZ LW1 ;DO 8 TIMES
02180 RET ;RET
02190 ERROR PUSH AF ;SAVE
02200 CALL 1C9H ;CLS
02210 POP AF ;GET ERROR
02220 CALL DISKER ;DIS
02230 LD HL,MESSO ;GET Q
02240 CALL 4467H ;DIS
02250 LD B,0 ;NO INPUT
02260 LD HL,INBUFF ;BUFFER
02270 CALL 40H ;INPUT & RET
02280 CALL 1C9H ;CLS
02290 RET ;RET
02300 MESSO DEFM 'HIT <ENTER> TO CONTINUE'
02310 DEFB 3
02320 LINEA LD B,16 ;#BYTES
02330 J5 LD A,(DE) ;GET BYTE
02340 INC DE ;BUMP
02350 CP 32 ;UNDER?
02360 JR C,J4 ;GO IF UNDIS
02370 CP 192 ;TO HFG
02380 JR NC,J4 ;UNDIS
02390 J6 LD (HL),A ;DISPLAY
02400 INC HL ;BUMP

```

Listing Continued...

... Continued Listing

02410	DJNZ	J5	;DO 16 TIMES
02420	RET		;RET
02430	J4 LD	A, '.'	;UNDIS
02440	JR	J6	;CONT
02450	;		
02460	KEYIN EQU	\$	
02470	CALL	49H	;GET KEY
02480	CP	','	;FOR?
02490	JR	Z, FOR	;GO IF SO
02500	CP	'-'	;BACK
02510	JR	Z, BACK	;GO IF BACK
02520	CP	1	;BREAK?
02530	JP	Z, J1	;ASK NEXT SEC
02540	CP	'M'	;MODIFY?
02550	JR	Z, MOD	;GO IF SO
02560	CP	'R'	;REREAD?
02570	JP	Z, REREAD	
02580	CP	'N'	;NEGATE?
02590	JP	Z, NEGATE	
02600	JR	KEYIN	;LOOP
02610	NEGATE LD	HL, BUFFER	;GET BUF
02620	LD	B, 0	;256 TIMES?
02630	N1 LD	A, (HL)	;GET BYTE
02640	NEG		
02650	LD	(HL), A	
02660	INC	HL	
02670	DJNZ	N1	
02680	CALL	PAGE	
02690	JR	KEYIN	
02700	FOR LD	HL, (SEC)	;GET SEC
02710	INC	HL	;ADD ONE
02720	FLX LD	(SEC), HL	;STORE
02730	JP	REREAD	;READ NEXT
02740	BACK LD	HL, (SEC)	;GET SEC
02750	DEC	HL	
02760	JR	FLX	
02770	;		
02780	;	MODIFY HANDLER	
02790	;		
02800	MOD EQU	\$	
02810	LD	(3C00H), A	;DISPLAY
02820	CALL	49H	;GET CHAR
02830	LD	(3C40H), A	;DIS
02840	CALL	CONV	;CONV
02850	JP	C, BADM	;GO IF BAD
02860	RLCA		
02870	RLCA		
02880	RLCA		
02890	RLCA		
02900	AND	0F0H	;PUT IN HI NIB
02910	PUSH	AF	;SAVE
02920	CALL	49H	;GET NEXT
02930	LD	(3C80H), A	;DIS
02940	CALL	CONV	;CONVERT
02950	JP	C, BADM	;GO IF ERROR

Listing Continued...

... Continued Listing

02960		LD	B,A	;GET #
02970		POP	AF	;GET FIRST
02980		OR	B	;ADD BITS
02990		LD	(POSN),A	;STORE POSN
03000		CALL	CLR	;CLR LEFT
03010	MOD1	CALL	POSNX	;POSN HL & DE
03020		CALL	INKEY	;GET INPUT FLASH
03030		CP	1	;BREAK
03040		JP	Z,REREAD	;GO IF ABORT MOD
03050		CP	8	;BACK
03060		JR	Z,BS	
03070		CP	9	;FOR
03080		JR	Z,FS	
03090		CP	13	;WRITE
03100		JR	Z,WRITE	;WRITE
03110		LD	(DE),A	;WRITE BYTE
03120		LD	A,(POSN)	;GET POSN
03130		INC	A	
03140		LD	(POSN),A	;INC IT
03150		CALL	PAGE	;DISPLAY
03160		JR	MOD1	;CONT
03170	BS	CALL	FF	
03180		LD	A,(POSN)	
03190		DEC	A	
03200	F9	LD	(POSN),A	;BACK
03210		JR	MOD1	;CONT
03220	FS	CALL	FF	
03230		LD	A,(POSN)	
03240		INC	A	
03250		JR	F9	;FORWARD
03260	FF	LD	A,(CUR)	
03270		LD	(HL),A	
03280		RET		
03290	MES90	DEFM	'WRITE MODIFIED SECTOR? (Y/N/P)'	
03300		DEFB	13	
03310	WRITE	CALL	1C9H	
03320		LD	HL,MES90	
03330		CALL	4467H	
03340	F70	CALL	49H	
03350		LD	DE,DCB	;I/O FCB
03360		CP	1	
03370		JP	Z,ASKS	
03380		CP	'N'	
03390		JP	Z,NO	
03400		CP	'Y'	
03410		JR	Z,FVV	
03420		CP	'P'	;PROT?
03430		JR	NZ,F70	;NO
03440		LD	A,(DE)	;GET
03450		OR	1	;WRITE PROT
03460		JR	SD3	;CONT
03470	FVV	LD	A,(DE)	;GET FCB BYTE
03480		RES	0,A	;PROT OFF
03490	SD3	LD	(DE),A	;XFR TO FCB
03500		LD	HL,BUFFER	

Listing Continued...

... Continued Listing

03510		LD	BC, (SEC)	
03520		CALL	1C9H	
03530		CALL	4442H	
03540		CALL	443CH	;WRITE & VERIFY
03550		JP	Z, JX1	;CONT
03560		CALL	ERROR	
03570		JP	ASKS	
03580	KEYLO	EQU	\$	
03590	BADM	CALL	CLR	;CLR LEFT
03600		JP	JX1	;GET SEC
03610	CLR	LD	HL, 3C00H	;GET SCREEN
03620		LD	DE, 64	;# TO CLR
03630		LD	B, 16	;# TO CLR
03640	B1	LD	(HL), 32	;CLR
03650		ADD	HL, DE	;BUMP NEXT
03660		DJNZ	B1	;DO 16 TIMES
03670		RET		;RET
03680	CONV	CP	'0'	;NON NUM?
03690		RET	C	;RET BAD
03700		CP	'G'	;NOPE
03710		JR	NC, BADC	;GO IF PAST
03720		CP	':'	;NUM?
03730		JR	C, GOODC	;GO IF SO
03740		CP	'A'	;BAD?
03750		JR	C, BADC	;GO IF SO
03760		SUB	55	;MAKE REAL
03770	C1	OR	A	;KILL C
03780		RET		;RET
03790	GOODC	SUB	48	;MAKE REAL
03800		JR	C1	;CONT
03810	BADC	SCF		;ERR
03820		RET		;RET
03830	POSNX	EQU	\$;POSITION HL, DE
03840		LD	A, (POSN)	;GET POSN
03850		PUSH	AF	;SAVE POSN
03860		LD	HL, 3C00H+48	;GET POSN START
03870		LD	DE, 64	;AMT TO ADD
03880	P1	SUB	16	;ON THIS LINE?
03890		JR	C, P2	;GO IF YES?
03900		ADD	HL, DE	;NEXT LINE
03910		JR	P1	;TRY AGAIN
03920	P2	AND	0FH	;GET LINE POSN
03930	P3	JR	Z, P4	;GO IF OK
03940		INC	HL	;BUMP
03950		DEC	A	;SUB1
03960		JR	P3	;RETRY
03970	P4	POP	AF	;GET POSN
03980		PUSH	HL	;SAVE POSN
03990		LD	HL, BUFFER	;GET IO START
04000		LD	E, A	;FOR OFFSET
04010		LD	D, 0	;CLR
04020		ADD	HL, DE	;GET BUFFER POSN
04030		EX	DE, HL	;PUT IN DE
04040		POP	HL	;GET SCREEN
04050		RET		;RET

Listing Continued...

... Continued Listing

04060	INKEY	EQU	\$;FLASHING INPUT
04070		LD	A, (HL)	;GET CHAR
04080		LD	(CUR), A	;STORE
04090	IKK	PUSH	DE	;SAVE
04100		CALL	2BH	;GET INPUT
04110		OR	A	;SET F
04120		POP	DE	;RESTOR
04130		RET	NZ	;IF OK
04140		LD	A, (COUNT)	;GEY COUNT
04150		INC	A	
04160		LD	(COUNT), A	;UPDATE
04170		CP	100	;FLASH?
04180		JR	NZ, IKK	;LOOP
04190		LD	A, (HL)	;GET SCREEN
04200		CP	143	;CUR?
04210		JR	Z, F1	;GO IF SO
04220		LD	(CUR), A	;STORE CHAR
04230		LD	(HL), 143	;CUR
04240	F2	XOR	A	
04250		LD	(COUNT), A	;RESET
04260		JR	IKK	
04270	F1	LD	A, (CUR)	;GET CHAR
04280		LD	(HL), A	;DISPLAY
04290		JR	F2	;CONT
04300	COUNT	NOP		
04310	CUR	NOP		
04320	NO	CALL	1C9H	
04330		CALL	PAGE	
04340		JP	KEYIN	
04350		END	5200H	

9

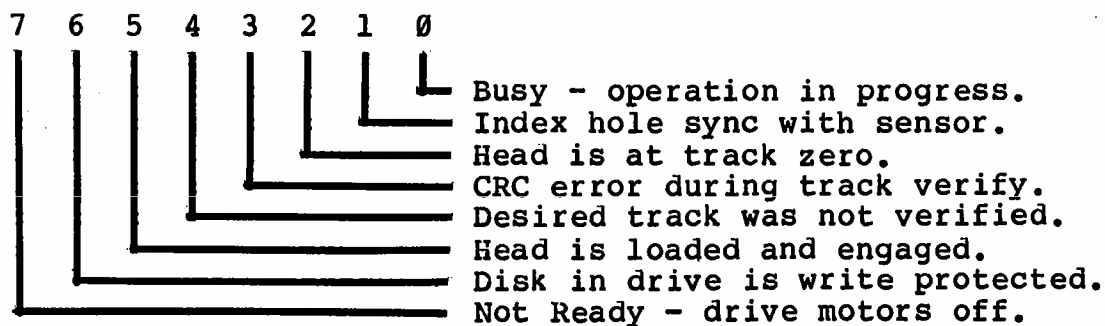
Miscellaneous Junk

Below is a quick-reference chart of the Western Digital FD1771 Floppy Disk Controller commands and status bits.

Figure 9.1 FD1771 Commands and Status Bits

Command Type	Command Name	Bits							
		7	6	5	4	3	2	1	0
1	Restore	0	0	0	0	H	V	r	r
1	Seek	0	0	0	1	H	V	r	r
1	Step	0	0	1	U	H	V	r	r
1	Step-in	0	1	0	U	H	V	r	r
1	Step-out	0	1	1	U	H	V	r	r
2	Read Sector	1	0	0	0	B	E	0	0
2	Write Sector	1	0	1	0	B	E	t	t
3	Read Address	1	1	0	0	0	E	0	0
3	Read Track	1	1	1	0	0	1	0	0
3	Write Track/Format	1	1	1	1	0	1	0	0
4	Force Interrupt/Reset	1	1	0	1	0	0	0	0

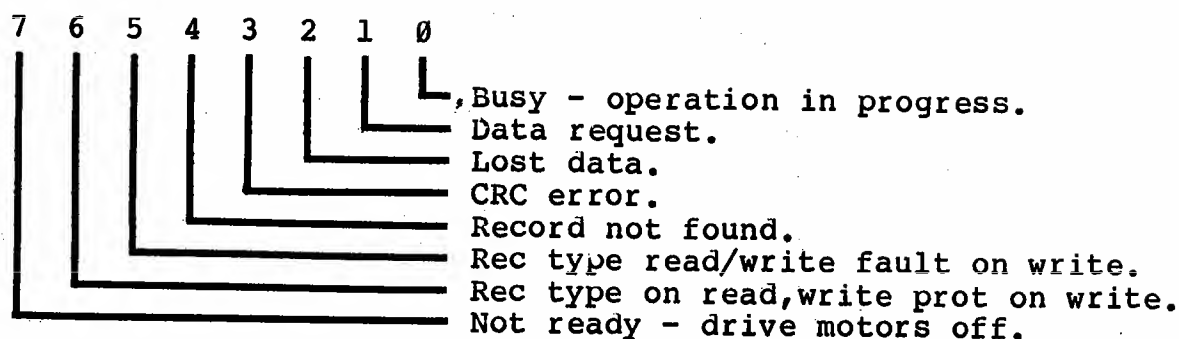
STATUS BIT DURING TYPE 1 COMMANDS.
Drive Select, Any Head Movement.



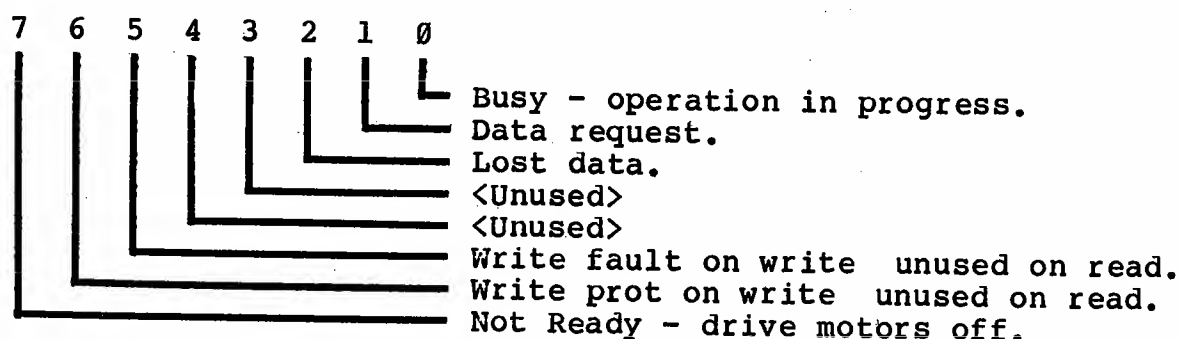
Listing Continued...

...Continued Listing

DURING TYPE 2 COMMANDS. Read Sector(s) / Write Sector(s)



DURING TYPE 3 COMMANDS. Write Track (Format), Read Track, Read Address.



Stepping Speeds of Popular Drives

Below is a list of the stepping speeds for most drives available for the TRS-80 Model I.

Figure 9.2 Drive Stepping Speeds

Brand	Max Stepping Speed.
Tandon (all track capacities)	6 ms.
MPI 40 track	6 ms.
Vista 40 track	12 ms.
Matchless 40 track	12 ms.
Micropolis 5/77 track	40 ms.
Shugart SA400	40 ms.
Tandy (older shugart SA400)	40 ms.

Small Disk Operating System (S/OS)

This chapter contains the small, but potentially powerful, disk operating system called 'S/OS' (which means Small Operating System). The S/OS system utilizes just about every disk and interrupt function described in this book. S/OS does not manage disk space, nor does it use or acknowledge a directory. But with a little programming on your end, this is quite possible. The main purpose of S/OS is to give you an idea on how a system is put together. S/OS may be used in special applications where all disk space is to be utilized. With S/OS you may access sectors on the disk in one of two ways: direct, i.e., you tell it in what drive, track and sector you want to access, or by the Relative Sector number from a given point (track and sector) on the disk.

Relative file offsets may be defined so that disk partitioning be logical to any programs you might use with S/OS. S/OS allows logical record lengths of 1 to 256 bytes per record. S/OS will automatically span sectors to fulfill a record access (when a record is spanned over two sectors). Also byte I/O routines via the ROM calls, 13 and 1B, are supported. 'File' access is done with File Control Blocks (FCB) just like a regular disk operating system. The FCB's in S/OS are 11 bytes long, and a technical description is discussed later.

S/OS is designed in such a way that various non-standard disk drivers may be assigned to one or more drives. This technical information is also explained later.

S/OS allows programs to enqueue an interrupt routine that can be executed by the interrupt processor every 25 milliseconds. There are 10 'slots' that may be defined or released by using various calls. These calls will be explained in the technical section.

S/OS was written using the ASSEM/80 editor/assembler, and is compatible with EDAS, and ALDS. All lines using DM for DEFM, and DB for DEFB may have to be modified in order to be assembled correctly by your assembler. Check your editor/assembler reference manual.

Technical Information Section

S/OS Call Vectors

S/OS is composed of two programs —

1. The S/OS Main Module.
2. The S/OS User Interface (this accepts keyboard input). All the disk I/O, FCB I/O, interrupt and special routines are contained in the S/OS main module. A number of routine entry vectors are used so that if you make changes to the S/OS main module, program compatibility will still exist.

S/OS does not use files like TRSDOS, but, as previously stated, S/OS has a way of doing a sort-of file I/O. File I/O in S/OS is simply done by using a FCB that

'points' to the file's first sector number (track and sector), then 'record' access is done by relative record number. The logical record length may be 1 to 256 bytes long. S/OS will span sectors to move a record, which allows maximum use of your disk space.

Here are the S/OS operation call vectors. Their respective vector addresses are in hex.

Disk I/O Primitives

Disk I/O primitives allow you to directly read and write to specified disk sectors. When calling these routines (disk I/O primitives) Register 'C' always contains the relative Drive Number in binary (*NOT ASCII*). Register pair 'HL' always points to the I/O Buffer (if required). Register 'D' contains the Track number, and 'E' contains the Sector number. Upon exit, Register 'A' always contains an error code. If the NZ flag is set, then 'A' contains an error code, otherwise no error occurred. This error return scheme is used by all routines that use Disk I/O in any way. All registers are intact upon return unless stated otherwise.

- 4600** Seek a drive's head to a desired track.
- Entry: C = drive number
 D = track number
- Exit: A = error code if NZ set
-
- 4603** Select a desired drive
- Entry: C = drive number
- Exit: The A register is altered.
 No error can occurred with this call.
-
- 4606** Verify that a drive is on-line, contains
 a diskette and the door is closed.
- Entry: C = drive number
- Exit: A = error code if NZ
-
- 4609** Read a disk sector
- Entry: C = drive number
 D = track number

Listing Continued...

... Continued Listing

```

                                E = sector number
                                HL -> 256 byte I/O buffer to put data

Exit:   A = error code if NZ set

460C   Read a disk ID address field.
        This routine is defined but is not implemented.

Entry:  C = drive number
        HL -> I/O buffer

Exit:   A = error code if NZ set

4612   Write a disk sector

Entry:  C = drive number
        D = track number
        E = sector number
        HL -> 256 byte I/O buffer that contains data

Exit:   A = error code if NZ set

4615   Write a read-protected disk sector

Entry:  C = drive number
        D = track number
        E = sector number
        HL -> 256 byte I/O buffer that contains data

Exit:   A = error code if NZ set

4618   Format a track of data
        This function is defined but not implemented

Entry:  C = drive number
        D = track number
        HL -> 256 byte I/O buffer that contains data

Exit:   A = error code if NZ set

461B   Initialized a desired drive
        This seeks the head to track 0 and syncs the DCT

Entry:  C = drive number

Exit:   A = error code if NZ set

```

S/OS File I/O Vector Calls

These calls are used to initiate, access and close a S/OS file.

461E Open a file control block (FCB).
 This creates a FCB that is used for doing file I/O.

Entry: C = drive number
 B = logical record length
 DE -> 11 byte FCB
 HL -> 256 byte sector I/O buffer

Exit: A = error code if NZ set

4621 Close a file control block (FCB).

This closes a file FCB. A close consists of writing any data in the buffer that has not been written to disk, such as when doing byte I/O. When doing byte I/O and record I/O with LRL < 256, bytes are Buffered so that the buffer isn't written until it's full, a POSN call is done or a CLOSE is done.

Entry: DE -> FCB

Exit: A = error code if NZ set

4624 Define the starting sector of a FCB

When a FCB is opened the starting records sector is initialized at track 0, sector 0. This Define call allows you to define the starting sector anywhere on the disk.

entry: DE -> FCB
 H = starting track
 L = starting sector

4627 Position the FCB for a record read.

entry: DE -> FCB
 BC = Logical record to access.

Listing Continued...

.. Continued Listing

462A Read the next record

This reads the next record currently in the FCB.
After the record is read, the FCB points to the next record.

entry: DE -> FCB
 HL -> Record buffer address. Not used if
 LRL = 256. Record buffer is NOT the
 256 byte sector buffer.

exit: A = error code if NZ set

462D Write the next record

This reads the next record currently in the FCB.
After the record is written, the FCB points to
the next record.

entry: DE -> FCB
 HL -> Record buffer address. Not used if
 LRL = 256. Record buffer is NOT the
 256 byte sector buffer.

exit: A = error code if NZ set

4630 Load an object file

This loads a machine-language program into memory.
The sectors, obviously, must be in sequential order.

entry: DE -> FCB

exit: HL -> transfer address (program start
 address)
 A = error code if NZ set

S/OS Special Function Calls**4633 Display an error code**

This will display the error code number in 'A' to the
video in this manner:

SYSTEM ERROR: 10

The number is in decimal.

Listing Continued...

... Continued Listing

entry: A = error code
exit: All registers altered (not primes).

4636 Display a line of text to the video.
4639 Send a line of text to the line printer.
463C Output a line to a device.

The text must be terminated by a 'ØD' which is transmitted or an Ø3 which is not transmitted.

entry: HL -> text to display or print.
DE -> DCB (463C call only).
exit: All registered altered (not primes).

See Microsoft BASIC decoded & other mysteries, published by IJG, Inc., for more information about the BASIC ROM device routine.

464B Enqueue an interrupt routine

These calls allow you to enqueue/dequeue routine(s) to be serviced every 25 milliseconds (40 times/second)

entry: A = slot to enqueue 0-9
HL -> routine address

464E Dequeue an interrupt routine

entry: A = slot to dequeue 0-9

The S/OS User Interface

The S/OS User Interface simply provides a way for you to enter three numbers, which are used as a drive, track and sector address where a machine-language 'file' is to be loaded. This allows invocation of any of your programs.

When S/OS is booted up, it will announce itself and prompt you for an input like so:

S/OS
?

Enter the drive, track and sector as shown below:

S/OS
? 0,5,0

The preceding would be interpreted as drive 0, track 5, sector 0. S/OS will attempt to load from this disk address. If it can't, an error will be displayed.

Byte

- 0** The drive number in binary
- 1** The drive's select code
- 2** drive status:

Bit

- 7** Undefined.
- 6** Undefined.
- 5** Dry select delay (1 =.5 sec vs. 1.0 sec).
- 4** Undefined.
- 3** Undefined.
- 2** Undefined.
- 1** Undefined.
- 0** Drive has been initialized.

- 3** Default directory track. Not currently used.
 - 4** The drive's current track number
 - 5** Drive's track capacity.
 - 6** The head step-rate bits. 0, 1, 2 or 3.
-

Drive Control Tables

Each drive has its own drive control table. This table contains information on each drive, such as its track capacity, its select code and other vital information.

The S/OS File Control Block

Here is a description of what the bytes in the file control block are used for:

Bytes 7, 8 and 9 make up a "relative byte address" in a file.

Byte

0	FCB Control Byte. 80H = file is open
1	Status Byte
7	Unused
6	Unused
5	Unused
4	Unused
3	Unused
2	Unused
1	A 1 means buff contains data to be written
0	A 1 means buff contain 'next' sector
2	LSB of 256-byte sector buffer address
3	MSB of 256-byte sector buffer address
4	Starting sector number of file
5	Starting track number of file
6	Drive number
7	Byte Offset within 'next' sector
8	LSB of 'next' sector
9	MSB of 'next' sector
10	Logical record length

```

*****
00001
*
00002 ;**      S/OS - SMALL/OPERATING SYSTEM      *
*
00003 ;**      VERSION 1.0 - MODEL I              *
*
00004 ;**      CREATED: 07/28/82                  *
*
00005 ;**      UPDATED: 08/06/82                  *
*
00006 ;**      (C) 1982 by Michael Wagner          *
*
00007
*****
00008 ;
00009 ;      Label equates
00010 ;
0000 7850 00011 @@@@@@ DW      EOP
37EF      00012 DATA  EQU    37EFH
37EC      00013 CMD    EQU    37ECH
37ED      00014 TRACK  EQU    37EDH
37EE      00015 SEC    EQU    37EEH
37E0      00016 SEL    EQU    37E0H
37E0      00017 INTRST EQU    37E0H
4500      00018 BUFF   EQU    4500H
00019 ;
402D      00020          ORG    402DH          ;VECTORS
00021 ;
402D C33B4F 00022 @JPDOS JP      JPDOS          ;ENTRY DIRECT
                                ;ENTRY
4030 C33B4F 00023 @ABORT JP      JPDOS          ;ABNORMAL ENTRY
4033 C3C94C 00024 @BYTIO JP      BYTEIO        ;DO BYTE I/O
00025 ;
400C      00026          ORG    400CH          ;INTERUPT VEC
00027 ;
400C C9     00028          RET                    ;BREAK VEC, RST28
400D 00     00029          NOP
400E 00     00030          NOP
400F C9     00031          RET                    ;DEBUG TRAP,
                                ;RST30
4010 00     00032          NOP
4011 00     00033          NOP
4012 C30049 00034          JP      INTER          ;INTER VEC, RST38
00035 ;
4600      00036          ORG    4600H          ;VECTOR AREA
00037 ;
4600 C3404A 00038 @SEEK  JP      SEEK          ;SEEK TRK
4603 C3444A 00039 @SELEC JP      SELECT        ;SELECT DRV
4606 C3484A 00040 @VALID JP      VALID        ;VALIDATE DRV
4609 C34C4A 00041 @RSEC  JP      RSEC         ;READ SEC
460C C3504A 00042 @RADR  JP      RADR         ;READ ADDRESS
460F C3544A 00043 @RCYL  JP      RCYL         ;READ TRK
4612 C3584A 00044 @WSEC  JP      WSEC         ;WRITE SEC
4615 C35C4A 00045 @WSECP JP      WSECP        ;WRITE SEC READ
                                ;PROT

```

Listing Continued...

... Continued Listing

4618	C3604A	00046	@FORMAT	JP	FORMAT	;FORMAT TRK
461B	C3644A	00047	@INITD	JP	INITD	;INIT DRV
		00048	;			
461E	C3194C	00049	@OPEN	JP	OPEN	;CREATE FCB
4621	C3544C	00050	@CLOSE	JP	CLOSE	;CLOSE FCB
4624	C36C4C	00051	@DEFIN	JP	DEFIN	;DEFINE STARTING
						;SEC
4627	C39B4C	00052	@POSN	JP	POSN	;POSN FCB TO
						;BYTE/SEC
462A	C3734D	00053	@READ	JP	READ	;READ NEXT RECORD
462D	C3974D	00054	@WRITE	JP	WRITE	;WRITE NEXT REC
4630	C3534E	00055	@LOAD	JP	LOAD	;LOAD FILE
		00056	;			
4633	C3FD4E	00057	@ERROR	JP	ERRORD	;ERROR DISP
4636	C3B94D	00058	@LINE	JP	LINE	;DISPLAY LINE
4639	C3B44D	00059	@PRINT	JP	PRINT	;PRINT LINE
463C	C3BC4D	00060	@DEV	JP	DEV	;DEVICE LINE
						;OUTPUT
463F	C30E4E	00061	@CI2	JP	CI2	;DEC CONV
4642	C3084E	00062	@CI3	JP	CI3	
4645	C3024E	00063	@CI4	JP	CI4	
4648	C3FC4D	00064	@CI5	JP	CI5	
464B	C3E64E	00065	@ENQUE	JP	ENQUE	
464E	C3F84E	00066	@DEQUE	JP	DEQUE	
		00067	;			
4700		00068		ORG	4700H	;PAGE 47 VARS
		00069	;			
4700	00	00070	TICKS	NOP		;REALTIME CLK
4701	00	00071	SECS	NOP		
4702	00	00072	MINS	NOP		
4703	00	00073	HOURS	NOP		
		00074	;			
4704	00	00075	YEAR	NOP		;DATE
4705	00	00076	DAY	NOP		
4706	00	00077	MONTH	NOP		
		00078	;			
4707	0052	00079	USER	DW	5200H	;USR MEM STA
4709	FFFF	00080	HIMEM	DW	0FFFFH	;MEM PROT PTR
470B	0000	00081		DW	0	;RESV
470D	6E49	00082	INTQUE	DW	RETINT	;INTERUPT QUEUES
470F	6E49	00083		DW	RETINT	
4711	6E49	00084		DW	RETINT	
4713	6E49	00085		DW	RETINT	
4715	6E49	00086		DW	RETINT	
4717	6E49	00087		DW	RETINT	
4719	6E49	00088		DW	RETINT	
471B	6E49	00089		DW	RETINT	
471D	6E49	00090		DW	RETINT	
471F	6E49	00091		DW	RETINT	
4721		00092	INTEND	EQU	\$	
		00093	;			
		00094	;			
		00095	;			
						VARIABLE TABLE
4800		00096		ORG	4800H	

Listing Continued ...

Continued Listing

```

00196 ;
00197 ;      DRIVE CONTROL TABLES
00198 ;
4A00 00199      ORG      4A00H
00200 ;
4A00 00201 DCTM      EQU,    4AH      ;MUST=MSB OF
                                         ;TABLE STA
00202 ;
4A00 104A 00203 DCT      DW      DCT0      ;DCT PTRS
4A02 1C4A 00204      DW      DCT1
4A04 284A 00205      DW      DCT2
4A06 344A 00206      DW      DCT3
4A08 0000 00207      DW      0      ;RESV FOR HIMEM
                                         ;DCTS
4A0A 0000 00208      DW      0      ;HARD DRIVES,
                                         ;ETC.
4A0C 0000 00209      DW      0
4A0E 0000 00210      DW      0
00211 ;
4A10 00212 DCT0      DM      0,1,0,17,0,39,3,10,24H,8
                                         00 01 00 11 00 27 03 0A 24 08
4A1A CF4A 00213      DW      DSKDVR
4A1C 00214 DCT1      DM      1,2,0,17,0,39,3,10,24H,8
                                         01 02 00 11 00 27 03 0A 24 08
4A26 CF4A 00215      DW      DSKDVR
4A28 00216 DCT2      DM      2,4,0,17,0,39,3,10,24H,8
                                         02 04 00 11 00 27 03 0A 24 08
4A32 CF4A 00217      DW      DSKDVR
4A34 00218 DCT3      DM      4,8,0,17,0,39,3,10,24H,8
                                         04 08 00 11 00 27 03 0A 24 08
4A3E CF4A 00219      DW      DSKDVR
00220 ;
00221 ;      DISK I/O ENTRY
00222 ;
4A40 3E01 00223 SEEK      LD      A,1      ;SEEK
4A42 1825 00224      JR      DSKHAN
4A44 3E02 00225 SELECT    LD      A,2      ;SELECT
4A46 1821 00226      JR      DSKHAN
4A48 3E03 00227 VALID     LD      A,3      ;VALIDATE
4A4A 181D 00228      JR      DSKHAN
4A4C 3E04 00229 RSEC      LD      A,4      ;READ SEC
4A4E 1819 00230      JR      DSKHAN
4A50 3E05 00231 RADR      LD      A,5      ;READ ADR
4A52 1815 00232      JR      DSKHAN
4A54 3E06 00233 RCYL      LD      A,6      ;READ CYL
4A56 1811 00234      JR      DSKHAN
4A58 3E07 00235 WSEC      LD      A,7      ;WRITE SEC
4A5A 180D 00236      JR      DSKHAN
4A5C 3E08 00237 WSECP     LD      A,8      ;WRITE SEC PROT
4A5E 1809 00238      JR      DSKHAN
4A60 3E09 00239 FORMAT    LD      A,9      ;FORMAT
4A62 1805 00240      JR      DSKHAN
4A64 3E0A 00241 INITD     LD      A,10     ;INIT DRV
4A66 C31F49 00242      JP      DSKINT

```

Listing Continued . . .

... Continued Listing

491F 1845	00143	DSKINT	JR	INTDUN	;DUN, RET
	00144				
4921	00145	INTDO	EQU	\$	
4921 C5	00146		PUSH	BC	;SAVE REGS
4922 D5	00147		PUSH	DE	
4923 E5	00148		PUSH	HL	
4924 214648	00149		LD	HL,CLK25	;GET 40MS COUNT
4927 35	00150		DEC	(HL)	;BUMP SEC?
4928 201F	00151		JR	NZ,INT04	;N
492A 3619	00152		LD	(HL),25	;RESTOR REF
492C 210147	00153		LD	HL,SECS	;GET SECS
492F 34	00154		INC	(HL)	;BMP SECS
4930 7E	00155		LD	A,(HL)	;GET VAL
4931 FE3C	00156		CP	60	;BMP MINS?
4933 2014	00157		JR	NZ,INT04	;N
4935 3600	00158		LD	(HL),0	;CLR SECS
4937 23	00159		INC	HL	;HL->MINS
4938 34	00160		INC	(HL)	;BMP MINS
4939 7E	00161		LD	A,(HL)	;GET MINS
493A FE3C	00162		CP	60	;BMP HOURS?
493C 200B	00163		JR	NZ,INT04	;N
493E 3600	00164		LD	(HL),0	;RESET MINS
4940 23	00165		INC	HL	;HL->HOURS
4941 34	00166		INC	(HL)	;BMP HOURS
4942 7E	00167		LD	A,(HL)	;GET HOURS
4943 FE18	00168		CP	24	;RESET HOURS?
4945 2002	00169		JR	NZ,INT04	;N
4947 3600	00170		LD	(HL),0	;NEW DAY
4949 210047	00171	INT04	LD	HL,TICKS	;TICKS COUNTER
494C 34	00172		INC	(HL)	;BMP MOD 256
494D 210D47	00173		LD	HL,INTQUE	;INT QUEUES
4950 7D	00174	INT05	LD	A,L	;GET L
4951 FE21	00175		CP	INTEND&255	;QUEUES DONE?
4953 280E	00176		JR	Z,INT06	;Y
4955 5E	00177		LD	E,(HL)	;GET ADR OF
4956 2C	00178		INC	L	;QUE
4957 56	00179		LD	D,(HL)	;MSB
4958 2C	00180		INC	L	
4959 E5	00181		PUSH	HL	;SAV NEXT QUE PTR
495A 216049	00182		LD	HL,INT07	;RET ADR
495D E5	00183		PUSH	HL	;PUSH
495E D5	00184		PUSH	DE	;PUSH QUE ADR FOR
					;JMP
495F C9	00185		RET		;JMP TO QUE
4960 E1	00186	INT07	POP	HL	;GET NEXT QUE PTR
4961 18ED	00187		JR	INT05	;CONT
4963 E1	00188	INT06	POP	HL	;RESTO REGS
4964 D1	00189		POP	DE	
4965 C1	00190		POP	BC	
4966 3AEC37	00191	INTDUN	LD	A,(37ECH)	;CLR FDC
4969 3AE037	00192		LD	A,(INTRST)	;RESET CLK
496C F1	00193		POP	AF	
496D FB	00194		EI		;ENABLE INT
496E C9	00195	RETINT	RET		;RET

Listing Continued...

... Continued Listing

```

00097 ;
4800 00098 INPBUF DS 64 ;DOSINP BUFF
4840 03 00099 DRIVES DB 3 ;# DRIVES IN
;SYSTEM
4841 0A 00100 RETRY DB 10 ;# I/O RETRIES
4842 00 00101 RTCNT NOP ;RE-TRY COUNTER
4843 0000 00102 DSKRET DW 0 ;DSK DVR ERR RET
;ADR
4845 00 00103 DSKRS NOP ;I/O IN PROGRESS
;FLAG
4846 19 00104 CLK25 DB 25 ;INT CLK SECONDS
;REF
4847 0000 00105 W1 DW 0 ;WORK REGS, TEMP
4849 0000 00106 W2 DW 0
484B 0000 00107 W3 DW 0
484D 0000 00108 W4 DW 0
484F 0000 00109 IOBUF DW 0 ;I/O BUFF PTR
4851 0000 00110 TKSEC DW 0 ;CYL & SEC
4853 00 00111 SPEED NOP ;SPEED MOD MULT
4854 03 00112 HIDRV DB 3 ;HIGHEST DRIVE #
4855 0000 00113 LINPAR DW 0 ;PTR TO RUN
;PARAMS
4857 00114 FCB DM 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
00 00 00 00 00 00 00 00 00 00 00 00 00
4863 0000 00115 LINPTR DW 0 ;PARAM PTR FOR
;EXEC
4865 00 00116 STAT1 NOP ;SYSSTAT
00117
;BIT0= CHECK
;AUTO IF 0

00118 ;
00119 ; INTERRUPT ROUTINE
00120 ;
4900 00121 ORG 4900H
00122 ;
4900 00123 INTER EQU $
4900 F5 00124 PUSH AF ;SAVE AF
4901 3AE037 00125 LD A,(INTRST) ;GET INT LATCH
4904 17 00126 RLA ;VALID INT?
4905 3018 00127 JR NC,DSKINT ;N
4907 17 00128 RLA ;DISK INT?
4908 3815 00129 JR C,DSKINT ;Y
00130 ;
490A 3A4548 00131 LD A,(DSKRS) ;I/O IN PROG?
490D B7 00132 OR A ;?
490E 2811 00133 JR Z,INTDO ;N
4910 CB7E 00134 BIT 7,(HL) ;DRIVE TIME OUT?
4912 2004 00135 JR NZ,INT01 ;Y
4914 CB46 00136 BIT 0,(HL) ;I/O DUN?
4916 2009 00137 JR NZ,INTDO ;N
4918 F1 00138 INT01 POP AF ;RESTO AF
4919 E3 00139 EX (SP),HL ;GET RET ADR
491A 2A4348 00140 LD HL,(DSKRET) ;GET ERROR RET
491D E3 00141 EX (SP),HL ;PUSH
491E F5 00142 PUSH AF ;SAVE AF

```

Listing Continued...

... Continued Listing

```

00243 ;
4A69      00244 DSKHAN EQU      $
4A69 CDA04A 00245 CALL      SAVREG      ;SAVE REGS ON STK
4A6C F5      00246 PUSH      AF          ;SAVE ENTRY CODE
4A6D CD7A4A 00247 CALL      GETDCT      ;GET DRIVE DCT
4A70 F1      00248 POP       AF          ;ENTRY CODE
4A71 E5      00249 PUSH      HL          ;SAVE HL (BUFF)
4A72 FD6E0A 00250 LD        L,(IY+10)    ;GET DRIVER ADR
4A75 FD660B 00251 LD        H,(IY+11)    ;MSB
4A78 E3      00252 EX        (SP),HL     ;SAVE DVR ADR,
                                           ;GET
                                           ;HL
4A79 C9      00253 RET                    ;JMP TO DVR
                                           00254 ;
                                           00255 ;
                                           00256 ;
                                           GET DRIVE DCT IN, IY
4A7A      00257 GETDCT EQU      $
4A7A E5      00258 PUSH      HL          ;SAVE HL
4A7B CB01    00259 RLC        C          ;DOUBLE C
4A7D 69      00260 LD        L,C        ;XFR TO L
4A7E 264A    00261 LD        H,DCTM     ;HL->DCT PTR FOR
                                           ;DRV
4A80 7E      00262 LD        A,(HL)     ;GET LSB OF DCT
                                           ;ADR
4A81 23      00263 INC        HL        ;BMP PTR
4A82 66      00264 LD        H,(HL)    ;GET MSB OF DCT
4A83 6F      00265 LD        L,A      ;HL->DCT
4A84 E5      00266 PUSH      HL        ;PASS TO IY
4A85 FDE1    00267 POP       IY
4A87 E1      00268 POP       HL
4A88 C9      00269 RET                    ;RESTO HL
                                           00270 ;
                                           00271 ;
                                           00272 ;
                                           SAVE REGS ON STACK
4A89      00273 PREP EQU      $
                                           ;SAVE REGS, CHK
4A89 E3      00274 EX        (SP),HL     ;FCB
                                           ;SAVE HL, GET RET
4A8A 229E4A 00275 LD        (JRET+1),HL ;ADR
4A8D E1      00276 POP       HL        ;STOR RET
4A8E 1A      00277 LD        A,(DE)    ;RESTOR HL
                                           ;GET FCB CTRL
4A8F 07      00278 RLCA              ;BYTE
4A90 3805    00279 JR        C,EX2     ;FCB OPEN?
4A92 3E11    00280 LD        A,17     ;YES, OK
4A94 B7      00281 OR        A        ;NOT OPEN ERR
4A95 1806    00282 JR        JRET     ;SET NZ
4A97 D5      00283 EX2    PUSH      DE ;RET
4A98 DDE1    00284 POP       IX      ;MAKE IX=FCB
4A9A CDA04A 00285 CALL      SAVREG     ;PUSH REGS ON
                                           ;STACK
4A9D C30000 00286 JRET      JP        0 ;RET TO CALLING
4AA0      00287 SAVREG EQU      $      ;RTN

```

Listing Continued...

... Continued Listing

4AA0	224748	00288	LD	(W1),HL	;SAVE HL
4AA3	E3	00289	EX	(SP),HL	;PUSH HL, GET RET
4AA4	22BD4A	00290	LD	(SRVEC+1),HL	;STO RET ADR
4AA7	D5	00291	PUSH	DE	;SAVE ALL REGS
4AA8	C5	00292	PUSH	BC	
4AA9	FDE5	00293	PUSH	IY	
4AAB	DDE5	00294	PUSH	IX	
4AAD	D9	00295	EXX		;PRIMES
4AAE	E5	00296	PUSH	HL	
4AAF	D5	00297	PUSH	DE	
4AB0	C5	00298	PUSH	BC	
4AB1	D9	00299	EXX		;RE-SWITCH
4AB2	08	00300	EX	AF,AF'	;SWI
4AB3	F5	00301	PUSH	AF	;SAVE AF
4AB4	08	00302	EX	AF,AF'	
4AB5	21BF4A	00303	LD	HL,REGRES	;RESTO RET ADR
4AB8	E5	00304	PUSH	HL	;PUSH FOR RET
4AB9	2A4748	00305	LD	HL,(W1)	;RESTO HL
4ABC	C30000	00306	JP	0	;RET TO CALLER
		00307 ;			
4ABF		00308	REGRES EQU	\$;RESTOR PUSHED
					;REGS
4ABF	08	00309	EX	AF,AF'	;GET PRIM
4AC0	F1	00310	POP	AF	;REST
4AC1	08	00311	EX	AF,AF'	
4AC2	D9	00312	EXX		;PRIMES
4AC3	C1	00313	POP	BC	
4AC4	D1	00314	POP	DE	
4AC5	E1	00315	POP	HL	
4AC6	D9	00316	EXX		
4AC7	DDE1	00317	POP	IX	
4AC9	FDE1	00318	POP	IY	
4ACB	C1	00319	POP	BC	
4ACC	D1	00320	POP	DE	
4ACD	E1	00321	POP	HL	
4ACE	C9	00322	RET		;RET TO ORG
					;CALLER
		00323 ;			
		00324 ;			
		00325 ;			
		00326	DSKDVR EQU	\$	
4ACF		00327	LD	(IOBUF),HL	;STO IO BUF
4ACF	224F48	00328	LD	(TKSEC),DE	;STO TK & SEC
4AD2	ED535148	00329	LD	HL,CMD	;HL->FDC CMD/STAT
4AD6	21EC37	00330	DEC	A	;SEEK?
4AD9	3D	00331	JR	Z,SEEK1	;Y
4ADA	2843	00332	DEC	A	;SELECT?
4ADC	3D	00333	JR	Z,SEL1	
4ADD	2862	00334	DEC	A	;VALIDATE
4ADF	3D	00335	JP	Z,VALID1	
4AE0	CAF94A	00336	DEC	A	;READ SEC?
4AE3	3D	00337	JP	Z,READ1	;Y
4AE4	CA864B	00338	DEC	A	;READ ADR
4AE7	3D	00339	RET	Z	
4AE8	C8				

Listing Continued...

... Continued Listing

4AE9 3D	00340		DEC	A	
4AEA C8	00341		RET	Z	;READ TRK
4AEB 3D	00342		DEC	A	
4AEC CA8D4B	00343		JP	Z,WRITE1	;WRITE SEC?
4AEF 3D	00344		DEC	A	
4AF0 CA944B	00345		JP	Z,PROT1	;WRITE PROT
4AF3 3D	00346		DEC	A	
4AF4 C8	00347		RET	Z	;FORMAT?
4AF5 3D	00348		DEC	A	
4AF6 286B	00349		JR	Z,INIT1	;INIT DRV?
4AF8 C9	00350		RET		;Y
	00351 ;				
	00352 ;				
	00353 ;				
	00354	VALID1	EQU	\$	
4AF9	00355		LD	(HL),0D0H	;FCD=MODE 1
4AFB CD414B	00356		CALL	SEL1	;SELECT DRV
4AFE 010020	00357		LD	BC,2000H	;TEST LENGTH
4B01 CD124B	00358	VD1	CALL	VDT	;TEST
4B04 20FB	00359		JR	NZ,VD1	;LOOP IF INDEX
					;HOLE
4B06 CD124B	00360	VD2	CALL	VDT	;TEST
4B09 28FB	00361		JR	Z,VD2	;LOOP IN NOT INX
4B0B CD124B	00362	VD3	CALL	VDT	;TEST
4B0E 20FB	00363		JR	NZ,VD3	;LOOP IF INX HOLE
4B10 AF	00364		XOR	A	;NO ERR
4B11 C9	00365		RET		
4B12 78	00366	VDT	LD	A,B	;GET COUNT
4B13 B1	00367		OR	C	;ERROR?
4B14 2804	00368		JR	Z,VTDE	;YES
4B16 0B	00369		DEC	BC	;DEC COUNT
4B17 CB4E	00370		BIT	1,(HL)	;RET WITH INDEX
					;STAT
4B19 C9	00371		RET		
4B1A F1	00372	VTDE	POP	AF	;KILL INT CALL
4B1B 3E15	00373		LD	A,21	;ERR
4B1D B7	00374		OR	A	;SET FLAGS
4B1E C9	00375		RET		
	00376 ;				
4B1F	00377	SEEK1	EQU	\$	
4B1F D5	00378		PUSH	DE	;SAVE TRK & SEC
4B20 CD634B	00379		CALL	INIT1	;INIT DRV
4B23 D1	00380		POP	DE	;GET TRK & SEC
4B24 CD414B	00381		CALL	SEL1	;SELECT
4B27 ED53EE37	00382		LD	(SEC),DE	;STO TRK & SEC
4B2B 7A	00383		LD	A,D	;GET TRK
4B2C FDBE04	00384		CP	(IY+4)	;ALREADY HERE?
4B2F C8	00385		RET	Z	;Y
4B30 3E10	00386		LD	A,10H	;SEEK W/VERF
4B32 FDB606	00387		OR	(IY+6)	;ADD STEP BITS
4B35 77	00388		LD	(HL),A	;ISSUE CMD
4B36 CD5E4B	00389	WAIT	CALL	DELAY	;DELAY FOR FDC
4B39 CD414B	00390	W00	CALL	SEL1	;SEL DRIVE
4B3C CB46	00391		BIT	0,(HL)	;DONE?

Listing Continued ...

... Continued Listing

4B3E 20F9	00392	JR	NZ,W00	;NO
4B40 C9	00393	RET		
4B41 7E	00394 SEL1	LD	A,(HL)	;GET STAT
4B42 F5	00395	PUSH	AF	;SAVE STAT
4B43 FD7E01	00396	LD	A,(IY+1)	;GET DRV SELCOD
4B46 32E037	00397	LD	(SEL),A	;SELECT
4B49 F1	00398	POP	AF	;GET PREV STAT
4B4A 07	00399	RLCA		;WERE DRVS ON?
4B4B D0	00400	RET	NC	;Y
4B4C C5	00401	PUSH	BC	;SAVE BC
4B4D 01608C	00402	LD	BC,8C60H	;DELAY COUNT
4B50 FDCB026E	00403	BIT	5,(IY+2)	;FULL SEC?
4B54 2002	00404	JR	NZ,S01	;Y
4B56 0646	00405	LD	B,46H	;1/2 SEC DELAY
4B58 CD6000	00406 S01	CALL	60H	;ROM DELAY RTN
4B5B C1	00407	POP	BC	;RES BC
4B5C 18E3	00408	JR	SEL1	;RESEL & RET
	00409 ;			
4B5E E3	00410 DELAY	EX	(SP),HL	;WASTE TIME FOR
				;FDC
4B5F E3	00411	EX	(SP),HL	
4B60 E3	00412	EX	(SP),HL	
4B61 E3	00413	EX	(SP),HL	
4B62 C9	00414	RET		
	00415 ;			
4B63 FDCB0246	00416 INIT1	BIT	0,(IY+2)	;DRV INIT?
4B67 C0	00417	RET	NZ	;Y
4B68 FDCB02C6	00418	SET	0,(IY+2)	;MAKE INIT
4B6C FD360400	00419	LD	(IY+4),0	;RESET TRK #
4B70 0660	00420	LD	B,96	;MAX TRK CNT
4B72 CD414B	00421 I00	CALL	SEL1	;SELECT DRIV
4B75 36D0	00422	LD	(HL),0D0H	;TYPE 1 MODE
4B77 CB56	00423	BIT	2,(HL)	;CYL 0?
4B79 C0	00424	RET	NZ	;DONE
4B7A 3E60	00425	LD	A,60H	;STEP OUT CMD
4B7C FDB606	00426	OR	(IY+6)	;ADD STEP RATE
4B7F 77	00427	LD	(HL),A	;STEP OUT
4B80 CD364B	00428	CALL	WAIT	;WAIT TIL DONE
4B83 10ED	00429	DJNZ	I00	;CONT
4B85 C9	00430	RET		
	00431 ;			
4B86 CD9B4B	00432 READ1	CALL	TASK	;DO FUNC
4B89 88031A02	00433	DB	88H,3,1AH,2	;CMD, STA ERRCOD
4B8D CD9B4B	00434 WRITE1	CALL	TASK	
4B90 A80B1200	00435	DB	0A8H,11,18,0	
4B94 CD9B4B	00436 PROT1	CALL	TASK	
4B97 A90B1200	00437	DB	0A9H,11,18,0	
	00438 ;			
4B9B E3	00439 TASK	EX	(SP),HL	;SAVE CMDREG, GET
PARS				
4B9C 7E	00440	LD	A,(HL)	;GET FDC CMDS
4B9D 32CF4B	00441	LD	(ISSUE+1),A	;STO
4BA0 23	00442	INC	HL	;BMP PTR
4BA1 7E	00443	LD	A,(HL)	;GET STA ERRCOD
4BA2 32094C	00444	LD	(ERROR+1),A	;STO

Listing Continued...

... Continued Listing

4BA5 23	00445	INC	HL	;BMP PTR
4BA6 7E	00446	LD	A, (HL)	;GET XFER OPS
4BA7 23	00447	INC	HL	
4BA8 66	00448	LD	H, (HL)	;MSB XFR OPS
4BA9 6F	00449	LD	L, A	;HL=XFER OPS
4BAA 22D64B	00450	LD	(XFER), HL	;STO XFER OPS
4BAD E1	00451	POP	HL	;HL=37ECH, CMD
4BAE 3A4148	00452	LD	A, (RETRY)	;GET # I/O
				;RETRIES
4BB1 324248	00453	LD	(RTCNT), A	;STO IN CNT
4BB4 CDBA4B	00454	CALL	REDO	;DO I/O
4BB7 7B	00455	LD	A, E	;GET ERRCOD
4BB8 B7	00456	OR	A	;SET FLAG
4BB9 C9	00457	RET		
	00458 ;			
4BBA 21EC37	00459 REDO	LD	HL, CMD	;RE-INIT CMD REG
4BBD ED5B5148	00460	LD	DE, (TKSEC)	;GET TK & SEC
4BC1 CD1F4B	00461	CALL	SEEK1	;SEEK IF NOT
4BC4 36D0	00462	LD	(HL), 0D0H	;RESET FDC
4BC6 ED4B4F48	00463	LD	BC, (IOBUF)	;GET I/O BUFF
4BCA 11EF37	00464	LD	DE, DATA	;DATA REG
4BCD F3	00465	DI		;KILL INT
4BCE 3600	00466 ISSUE	LD	(HL), 0	;ISSUE CMD
4BD0 CD5E4B	00467	CALL	DELAY	;DELAY FOR FDC
4BD3 0A	00468 LOOP	LD	A, (BC)	;GET BUF CHR
4BD4 1804	00469	JR	DIO	;CONT
4BD6 00	00470 XFER	NOP		;XFER OPS HERE
4BD7 00	00471	NOP		
4BD8 03	00472	INC	BC	;BMP BUFF
4BD9 0A	00473	LD	A, (BC)	;GET NEXT BUF CHR
4BDA CB4E	00474 DIO	BIT	1, (HL)	;DRQ?
4BDC 20F8	00475	JR	NZ, XFER	
4BDE CB46	00476	BIT	0, (HL)	;DUN?
4BE0 2808	00477	JR	Z, IODUN	
4BE2 CB4E	00478	BIT	1, (HL)	;DRQ?
4BE4 20F0	00479	JR	NZ, XFER	
4BE6 CB7E	00480	BIT	7, (HL)	;TIME OUT?
4BE8 28F0	00481	JR	Z, DIO	;N
4BEA	00482 IODUN	EQU	\$	
4BEA 7E	00483	LD	A, (HL)	;GET FDC STAT
	00484 ;	EI		;ENABLE INT
4BEB 47	00485	LD	B, A	;SAVE ERR COD
4BEC 07	00486	RLCA		;TIME OUT?
4BED 1E15	00487	LD	E, 21	;TIME OUT ERRCOD
4BEF D8	00488	RET	C	;Y
4BF0 07	00489	RLCA		;WR PROT/SEC TYP?
4BF1 3815	00490	JR	C, ERROR	
4BF3 07	00491	RLCA		;WR FAULT/SEC
				;TYP?
4BF4 3812	00492	JR	C, ERROR	
4BF6 78	00493	LD	A, B	;GET FDC STAT
4BF7 1E00	00494	LD	E, 0	;INIT NO ERR
4BF9 E61C	00495	AND	1CH	;ERR?
4BFB C8	00496	RET	Z	;N

Listing Continued...

... Continued Listing

4BFC 214248	00497	LD	HL, RTCNT	; RETRY CNT
4BFF 35	00498	DEC	(HL)	; RETRY?
4C00 2806	00499	JR	Z, ERROR	; NO, ERR
4C02 FDCB0286	00500	RES	0, (IY+2)	; FORCE INIT
4C06 18B2	00501	JR	REDO	; REDO
	00502 ;			
4C08 1E00	00503 ERROR	LD	E, 0	; ERRCOD START
4C0A 0F	00504	RRCA		
4C0B 0F	00505	RRCA		
4C0C 0F	00506	RRCA		
4C0D D8	00507	RET	C	; LOST DATA
4C0E 1C	00508	INC	E	
4C0F 0F	00509	RRCA		
4C10 D8	00510	RET	C	; PARITY ERROR
4C11 1C	00511	INC	E	
4C12 0F	00512	RRCA		
4C13 D8	00513	RET	C	; RECORD NOT FOUND
4C14 1C	00514	INC	E	
4C15 0F	00515	RRCA		
4C16 D8	00516	RET	C	; WR FAULT/SEC
				; TYP=6
4C17 1C	00517	INC	E	; MUST BE
4C18 C9	00518	RET		; WR PROT/SEC
				; TYP=7
	00519 ;			
	00520 ;			
	00521 ;			
		FCB HANDLERS		
4C19	00522 OPEN	EQU	\$; OPEN FCB
4C19 CDA04A	00523	CALL	SAVREG	; SAVE REGS
4C1C D5	00524	PUSH	DE	; GET FCB IN IX
4C1D DDE1	00525	POP	IX	; IX->FCB
4C1F 4F	00526	LD	C, A	; XFER DRIV #
4C20 3A5448	00527	LD	A, (HIDRV)	; GET HIGH DRV #
4C23 B9	00528	CP	C	; BAD DRIVE?
4C24 382A	00529	JR	C, EX1	; YES
4C26 DD360080	00530	LD	(IX), 80H	; INIT CTRL BYTE
4C2A DD360100	00531	LD	(IX+1), 0	; CLR STATUS BYTE
4C2E DD7502	00532	LD	(IX+2), L	; STOR LSB OF I/O
				; BUFF
4C31 DD7403	00533	LD	(IX+3), H	; STOR MSB OF BUFF
4C34 DD360400	00534	LD	(IX+4), 0	; STARTING SEC,
				; TRK=0, 0
4C38 DD360500	00535	LD	(IX+5), 0	; (SEE @DEFIN
				; CALL)
4C3C DD7106	00536	LD	(IX+6), C	; STOR DRIVE
4C3F DD360700	00537	LD	(IX+7), 0	; CLR 'NEXT REC'
				; FIELD
4C43 DD360800	00538	LD	(IX+8), 0	
4C47 DD360900	00539	LD	(IX+9), 0	
4C4B DD700A	00540	LD	(IX+10), B	; STORE LOGICAL
REC				
				; LEN
4C4E AF	00541	XOR	A	; NO ERROR
4C4F C9	00542	RET		; RET
4C50 3E10	00543 EX1	LD	A, 16	; BAD DRIVE ERR

Listing Continued...

... Continued Listing

4C52 B7	00544	OR	A		
4C53 C9	00545	RET			;SET NZ
	00546 ;				
	00547 ;				
	00548 ;			CLOSE AN FCB	
4C54	00549	CLOSE	EQU	\$	
4C54 CD894A	00550	CALL		PREP	
4C57 CD604C	00551	CALL		FLUSH	;CHECK IF FCB OK
					;WRITE UN-WRITE
					;DATA (IF ANY)
4C5A C0	00552	RET	NZ		;RET IF ERR
4C5B DD360000	00553	LD		(IX),0	;FCB CTRL
					;BYTE= OFF
4C5F C9	00554	RET			;RET
	00555 ;				
	00556 ;				
	00557 ;				
	00558			WRITE UN-WRITTEN DATA	
4C60	00558	FLUSH	EQU	\$	
4C60 DDCB014E	00559	BIT		1,(IX+1)	;ANY DATA TO
					;WRITE?
4C64 C8	00560	RET	Z		;NO, RET
4C65 DDCB018E	00561	RES		1,(IX+1)	;RESET FLAG
4C69 C33C4D	00562	JP		SWRITE	;WRITE REL SEC
	00563 ;				
	00564 ;				
	00565 ;				
	00566			DEFINE STARTING SECTOR IN FCB	
4C6C	00566	DEFIN	EQU	\$	
4C6C C5	00567	PUSH		BC	;SAVE BC
4C6D 010000	00568	LD		BC,0	;POSN FCB TO
					;REC 0
4C70 CD2746	00569	CALL		@POSN	;ALSO FLUSHES
					;BUFF
4C73 C1	00570	POP		BC	;RESTO BC
4C74 C0	00571	RET	NZ		;ERROR
4C75 CD894A	00572	CALL		PREP	;SAVE REGS,
					;IX=FCB
4C78 DD4E06	00573	LD		C,(IX+6)	;GET DRIV
4C7B CD7A4A	00574	CALL		GETDCT	;GET DCT OF DRV
4C7E FD7E05	00575	LD		A,(IY+5)	;GET # TRKS
4C81 BC	00576	CP		H	;TRK SPEC OK?
4C82 3813	00577	JR		C,EX3	;NO, ERR
4C84 DD7405	00578	LD		(IX+5),H	;STO TRK IN FCB
4C87 FD7E07	00579	LD		A,(IY+7)	;GET SECS/TRK
4C8A E67F	00580	AND		127	;MASK VALUE
4C8C BD	00581	CP		L	;SEC SPEC OK?
4C8D 3805	00582	JR		C,EX4	;NO, ERR
4C8F DD7504	00583	LD		(IX+4),L	;STOR IN FCB
4C92 AF	00584	XOR		A	;NO ERR
4C93 C9	00585	RET			
4C94 3E13	00586	LD		A,19	;BAD SEC ERR
4C96 01	00587	DEFB		1	;SAVE A
					;(LD BC,NN)
4C97 3E12	00588	LD		A,18	;BAD TRK ERR
4C99 B7	00589	OR		A	;SET NZ
4C9A C9	00590	RET			
	00591 ;				
	00592 ;				
				POSN FCB TO RECORD	

Listing Continued...

Continued Listing

```

00593 ;
4C9B      00594 POSN    EQU      $
4C9B CD894A 00595      CALL     PREP          ;VERF FCB
4C9E CD604C 00596      CALL     FLUSH        ;FLUSH BUFF IF
                                           ;NEEDED
4CA1 DD7E0A 00597      LD       A,(IX+10)    ;GET LOG REC LEN
4CA4 1E00   00598      LD       E,0          ;DE=LRL
4CA6 B7     00599      OR       A            ;LRL=256?
4CA7 2815   00600      JR       Z,PO1        ;YES, BC=REL
                                           ;SEC #
4CA9 AF     00601      XOR      A            ;A=0
4CAA 210000 00602      LD       HL,0         ;HL=SEC 0
4CAD 04     00603 PO2   INC      B            ;MSB REC # 0?
4CAE 05     00604      DEC      B            ;?
4CAF 2004   00605      JR       NZ,PO3       ;NO
4CB1 0C     00606      INC      C            ;LSB REC # 0?
4CB2 0D     00607      DEC      C            ;?
4CB3 2807   00608      JR       Z,PO4        ;POSN DONE
4CB5 83     00609 PO3   ADD      A,E         ;ADD LRL
4CB6 0B     00610      DEC      BC          ;DEC REC CNT
4CB7 30F4   00611      JR       NC,PO2       ;GO IF NC
4CB9 23     00612      INC      HL          ;BMP REL SEC CNT
4CBA 18F1   00613      JR       PO2         ;CONT
4CBC E5     00614 PO4   PUSH     HL          ;PASS REL SEC TO
                                           ;BC
4CBD C1     00615      POP      BC          ;
4CBE DD7108 00616 PO1   LD       (IX+8),C    ;STOR LSB OF SEC
4CC1 DD7009 00617      LD       (IX+9),B    ;MSB OF SECTOR
4CC4 DD7707 00618      LD       (IX+7),A    ;STOR BYTE WITHIN
                                           ;SEC
4CC7 AF     00619      XOR      A            ;NO ERROR
4CC8 C9     00620      RET
00621 ;
00622 ;      HANDLE BYTE I/O VIA 13H, 1BH ROM CALLS
00623 ;
4CC9      00624 BYTEIO EQU      $
4CC9 CDA04A 00625      CALL     SAVREG        ;SAV REGS
4CCC DDCB007E 00626     BIT      7,(IX)      ;FCB?
4CD0 C8     00627      RET      Z            ;NO, RET
4CD1 05     00628      DEC      B            ;READ?
4CD2 CAF04C 00629      JP       Z,BYTEI      ;READ BYTE
00630 ;
00631 ;      WRITE A BYTE
00632 ;
4CD5      00633 BYTEO  EQU      $
4CD5 C5     00634      PUSH     BC          ;C=BYT
                                           ;
4CD6 CD224D 00635      CALL     SREAD        ;MAKE BUFF=SEC
4CD9 C1     00636      POP      BC          ;C=BYTE
4CDA C0     00637      RET      NZ          ;GO IF SREAD ERR
4CDB 71     00638      LD       (HL),C      ;XFR BYT TO BUFF
4CDC DDCB01CE 00639     SET      1,(IX+1)    ;'BUFF=O/S DATA'
4CE0 DD3407 00640      INC      (IX+7)      ;BMP CUR BYTE #
4CE3 2802   00641      JR       Z,BTX      ;GO IF TIME TO
                                           ;WRITE

```

Listing Continued...

... Continued Listing

4CE5	AF	00642		XOR	A		;NO ERR
4CE6	C9	00643		RET			
4CE7	CD604C	00644	BTX	CALL	FLUSH		;WRITE BUFF
4CEA	C0	00645		RET	NZ		;ERROR
4CEB	CD164D	00646		CALL	INCSEC		;BMP SEC #
4CEE	AF	00647		XOR	A		;NO ERR
4CEF	C9	00648		RET			
		00649					
		00650					
		00651					
4CF0		00652	BYTEI	EQU	\$		
4CF0	CD604C	00653		CALL	FLUSH		;FLUSH BUFF IF
							;NEEDED
4CF3	CD224D	00654		CALL	SREAD		;READ NEXT SEC
4CF6	C0	00655		RET	NZ		;ERR
4CF7	CD004D	00656	BI0	CALL	PTHL		;HL->NEXT BYT
4CFA	CD124D	00657		CALL	INCBYT		;BMP FCB 'NEXT
							;BYTE'
4CFD	AF	00658		XOR	A		;SET ZERO FLAG
4CFE	7E	00659		LD	A, (HL)		;GET BYTE
							;FINALLY!
4CFF	C9	00660		RET			;RET
		00661					
4D00	CD0B4D	00662	PTHL	CALL	PTH		;HL->BUFFER
4D03	DD7E07	00663		LD	A, (IX+7)		;GET CUR BYTE
4D06	85	00664		ADD	A, L		;ADD CUR BYT
4D07	6F	00665		LD	L, A		;XFER
4D08	D0	00666		RET	NC		;RET IF HL->BYTE
4D09	24	00667		INC	H		;BMP MSB,
							;COMPEN-
							;SATE FOR
							;CROSSING
							;PAGE BOUNDRIES
4D0A	C9	00668					
4D0B	DD6E02	00669		RET			
4D0E	DD6603	00670	PTH	LD	L, (IX+2)		;GET LSB BUFF PTR
4D11	C9	00671		LD	H, (IX+3)		;GET MSB
		00672		RET			
		00673					
4D12	DD3407	00674	INCBYT	INC	(IX+7)		;BMP NEXT BYT
4D15	C0	00675		RET	NZ		;RET IF OK
4D16	DDCB0186	00676	INCSEC	RES	0, (IX+1)		;RES 'BUFF=SEC'
							;FLAG
4D1A	DD3408	00677		INC	(IX+8)		;BMP NEXT SEC
4D1D	C0	00678		RET	NZ		;RET IF NO MSB
							;BMP
4D1E	DD3409	00679		INC	(IX+9)		;BMP MSB OF NEXT
							;SEC
4D21	C9	00680		RET			
		00681					
		00682					
		00683					
		00684					
4D22		00685	SREAD	EQU	\$		

Listing Continued...

... Continued Listing

4D22	DDCB0146	00686	BIT	0,(IX+1)	;BUFF = SECTOR?
4D26	2802	00687	JR	Z,SS1	;NO
4D28	AF	00688	XOR	A	;NO ERR
4D29	C9	00689	RET		
4D2A	CD4F4D	00690	CALL	COMP	;COMPUTE TRK &
					;SEC
4D2D	DD4E06	00691	LD	C,(IX+6)	;GET DRIVE
4D30	CD0B4D	00692	CALL	PTH	;HL->BUFF
4D33	CD0946	00693	CALL	@RSEC	;READ SEC
4D36	C0	00694	RET	NZ	;ERR
4D37	DDCB01C6	00695	SET	0,(IX+1)	;SET 'BUFF=SEC'
4D3B	C9	00696	RET		;RET
		00697 ;			
		00698 ;	WRITE	THE FCB'S 'NEXT'	SECTOR
		00699 ;			
4D3C		00700	SWRITE	EQU	\$
4D3C	CD4F4D	00701	CALL	COMP	;COMPUTE TRK &
					;SEC
4D3F	DD4E06	00702	LD	C,(IX+6)	;GET DRIV
4D42	CD0B4D	00703	CALL	PTH	;HL->BUFFER
4D45	DDCB0156	00704	BIT	2,(IX+1)	;WRITE SEC PROT?
4D49	CA1246	00705	JP	Z,@WSEC	;NO
4D4C	C31546	00706	JP	@WSECP	;YES
		00707 ;			
		00708 ;	COMPUTE	REAL TRK & SEC	FROM REL SEC
		00709 ;			
4D4F		00710	COMP	EQU	\$
4D4F	DD4E06	00711	LD	C,(IX+6)	;GET DRIV
4D52	CD7A4A	00712	CALL	GETDCT	;GET DCT
4D55	DD5E04	00713	LD	E,(IX+4)	;STA SECTOR
4D58	DD5605	00714	LD	D,(IX+5)	;STA TRK
4D5B	DD4E08	00715	LD	C,(IX+8)	;GET REL SEC TO
					;COMP
4D5E	DD4609	00716	LD	B,(IX+9)	
4D61	78	00717	LD	A,B	;DONE?
4D62	B1	00718	OR	C	;?
4D63	C8	00719	RET	Z	;YES, DE=TRK &
					;SEC
4D64	0B	00720	DEC	BC	;DEC REC SEC
					;COUNT
4D65	1C	00721	INC	E	;BMP SEC #
4D66	FD7E07	00722	LD	A,(IY+7)	;GET SEC #
4D69	E67F	00723	AND	127	;GET # SEC/TRK
4D6B	BB	00724	CP	E	;TIME TO BUMP TRK
					;#?
4D6C	20F3	00725	JR	NZ,CO1	;NO
4D6E	1E00	00726	LD	E,0	;RESET SECTOR CNT
4D70	14	00727	INC	D	;BMP TRK
4D71	18EE	00728	JR	CO1	;CONT
		00729 ;			
		00730 ;	READ	NEXT RECORD	
		00731 ;			
4D73		00732	READ	EQU	\$
4D73	CD894A	00733	CALL	PREP	;VERF FCB
4D76	DD7E0A	00734	LD	A,(IX+10)	;GET LEL
4D79	B7	00735	OR	A	;LRL=256?

Listing Continued...

... Continued Listing

4D7A 280B	00736	JR	Z,XR01	;Y
4D7C 47	00737	LD	B,A	;GET LRL
4D7D CD1300	00738 XR02	CALL	13H	;READ BYTE
4D80 C0	00739	RET	NZ	;ERR
4D81 77	00740	LD	(HL),A	;STOR IN USEREC
4D82 23	00741	INC	HL	;BMP HL
4D83 10F8	00742	DJNZ	XR02	;LOOP
4D85 AF	00743	XOR	A	;NO ERR
4D86 C9	00744	RET		
	00745 ;			
4D87 CD604C	00746 XR01	CALL	FLUSH	;FLUSH BUFF
4D8A DDCB0186	00747	RES	0,(IX+1)	;BUFF<>NEXT
4D8E CD224D	00748	CALL	SREAD	;READ NEXT SEC
4D91 C0	00749	RET	NZ	;ERR
4D92 CD164D	00750	CALL	INCSEC	;BMP SEC
4D95 AF	00751	XOR	A	;NO ERR
4D96 C9	00752	RET		
	00753 ;			
	00754 ;	WRITE NEXT REC		
	00755 ;			
4D97	00756 WRITE	EQU	\$	
4D97 CD894A	00757	CALL	PREP	;VERF FCB
4D9A DD7E0A	00758	LD	A,(IX+10)	;GET LRL
4D9D B7	00759	OR	A	;LRL=256?
4D9E 280B	00760	JR	Z,WRI	;Y
4DA0 47	00761	LD	B,A	;B=LRL
4DA1 7E	00762 WS00	LD	A,(HL)	;GET NEXT CHR
4DA2 CD1B00	00763	CALL	1BH	;WRITE BYT
4DA5 C0	00764	RET	NZ	;ERROR
4DA6 23	00765	INC	HL	;BMP BUFPTR
4DA7 10F8	00766	DJNZ	WS00	;LOOP
4DA9 AF	00767	XOR	A	;NO ERR
4DAA C9	00768	RET		
	00769 ;			
4DAB	00770 WRI	EQU	\$	
4DAB CD3C4D	00771	CALL	SWRITE	;WRITE BUFFER
4DAE C0	00772	RET	NZ	;ERROR
4DAF CD164D	00773	CALL	INCSEC	;BMP SEC
4DB2 AF	00774	XOR	A	;NO ERR
4DB3 C9	00775	RET		
	00776 ;			
	00777 ;	DISP/PRINT LINE		
	00778 ;			
4DB4 112540	00779 PRINT	LD	DE,4025H	;DCB ADR
4DB7 1803	00780	JR	DEV	;CON
4DB9 111D40	00781 LINE	LD	DE,401DH	;DCB ADR
4DBC 7E	00782 DEV	LD	A,(HL)	;GET CHR
4DBD FE03	00783	CP	3	;EOL?
4DBF C8	00784	RET	Z	;Y
4DC0 CD1B00	00785	CALL	1BH	;WRITE BYTE TO
				;DEV
4DC3 7E	00786	LD	A,(HL)	;GET BYT
4DC4 FE0D	00787	CP	13	;EOL?
4DC6 C8	00788	RET	Z	;Y

Listing Continued...

... Continued Listing

```

4DC7 23          00789          INC      HL          ;BMP LINPTR
4DC8 18F2        00790          JR        DEV        ;CONT
                00791 ;
4DCA             00792 @INIT    DM          28,31,'S/OS disk operating
system,ver 1.0 08/04/82',10,13
                1C 1F 53,2F 4F 53 20 64 69 73 6B 20
                6F 70 65 72 61 74 69 6E 67 20 73 79
                73 74 65 6D 2C 20 76 65 72 20 31 2E
                30 20 30 38 2F 30 34 2F 38 32 0A 0D
                00793 ;
                00794 ;          PUTS DECIMAL EQU OF (INT) VALUE.
                00795 ;          ENTRY : HL->DEST, C=0 LEADING ZEROS
                00796 ;
4DFA 0000        00797 INT      DEFW      0
4DFC 111027      00798 CI5     LD         DE,10000
4DFF CD194E      00799          CALL      CVI
4E02 11E803      00800 CI4     LD         DE,1000
4E05 CD194E      00801          CALL      CVI
4E08 116400      00802 CI3     LD         DE,100
4E0B CD194E      00803          CALL      CVI
4E0E 110A00      00804 CI2     LD         DE,10
4E11 CD194E      00805          CALL      CVI
4E14 3AFA4D      00806          LD        A,(INT)
4E17 1827        00807          JR        LOP4
4E19 C5          00808 CVI     PUSH      BC
4E1A 0600        00809          LD        B,0
4E1C E5          00810          PUSH     HL
4E1D 2AFA4D      00811          LD        HL,(INT)
4E20 B7          00812 LOP     OR        A
4E21 ED52        00813          SBC      HL,DE
4E23 3807        00814          JR        C,LOP3
4E25 F5          00815          PUSH     AF
4E26 04          00816          INC      B
4E27 F1          00817          POP      AF
4E28 2803        00818          JR        Z,LOP2
4E2A 18F4        00819          JR        LOP
4E2C 19          00820 LOP3    ADD      HL,DE
4E2D 22FA4D      00821 LOP2    LD        (INT),HL
4E30 78          00822          LD        A,B
4E31 E1          00823          POP      HL
4E32 C1          00824          POP      BC
4E33 B7          00825          OR        A
4E34 2804        00826          JR        Z,LOP4A
4E36 0E00        00827          LD        C,0
4E38 1806        00828          JR        LOP4
4E3A 79          00829 LOP4A   LD        A,C
4E3B B7          00830          OR        A
4E3C 2802        00831          JR        Z,LOP4
4E3E 3EF0        00832          LD        A,0FH
4E40 C630        00833 LOP4    ADD      A,30H
4E42 77          00834          LD        (HL),A
4E43 23          00835          INC      HL
4E44 C9          00836          RET
                00837 ;
4E45 7E          00838 NEXT   LD        A,(HL)          ;GET CHR

```

Listing Continued...

... Continued Listing

4E46 FE0D	00839	CP	13	;C/R? EOL?
4E48 C8	00840	RET	Z	;Y
4E49 FE2C	00841	CP	' , '	;COMMA?
4E4B 2803	00842	JR	Z,NXX	;Y, IGNOR
4E4D FE20	00843	CP	32	;SPACE?
4E4F C0	00844	RET	NZ	;NO, RET
4E50 23	00845 NXX	INC	HL	;IGNOR SPAC
4E51 18F2	00846	JR	NEXT	;LOOP
	00847 ;			
	00848 ;			
	00849 ;			
		LOAD	A OBJ PROGRAM, DE=TK, SEC, C=DRIVE	
4E53	00850 LOAD	EQU	\$	
4E53 CD5B4E	00851	CALL	LODX	;LOAD AREA
4E56 C0	00852	RET	NZ	;ERR
4E57 2A4B48	00853	LD	HL, (W3)	;GET XFR ADR
4E5A C9	00854	RET		
	00855 ;			
4E5B CDA04A	00856 LODX	CALL	SAVREG	;SAV REGS
4E5E CD0646	00857	CALL	@VALID	;VAL DRIVE
4E61 C0	00858	RET	NZ	;DRIVE NOT READY
4E62 CD7A4A	00859	CALL	GETDCT	;IY=DCT
4E65 210045	00860	LD	HL,BUFF	;INP BUFF
4E68 D9	00861	EXX		;SW REGS
4E69 CDBB4E	00862 LMAN	CALL	GB	;GET BYT
4E6C B7	00863	OR	A	;COMM?
4E6D 2841	00864	JR	Z,LCOM	;Y
4E6F FE01	00865	CP	1	;LOAD BLK?
4E71 2021	00866	JR	NZ,LXFR	;NO
4E73 CDBB4E	00867	CALL	GB	;GET BLK LEN
4E76 D602	00868	SUB	2	;GET MEM BLK
4E78 47	00869	LD	B,A	;STO IN CNT
4E79 CDBB4E	00870	CALL	GB	;GET LD ADR LSM
4E7C 6F	00871	LD	L,A	;XFR
4E7D CDBB4E	00872	CALL	GB	;GET MSB
4E80 67	00873	LD	H,A	;XFR MSB
4E81 CDBB4E	00874 LMOV	CALL	GB	;GET OBJ BYT
4E84 77	00875	LD	(HL),A	;STO IN MEM
4E85 BE	00876	CP	(HL)	;VERF LD
4E86 2005	00877	JR	NZ,LER1	;BAD MEM
4E88 23	00878	INC	HL	;BMP ADR
4E89 10F6	00879	DJNZ	LMOV	;LOOP
4E8B 18DC	00880	JR	LMAN	;DO NEXT BLK
4E8D 3E16	00881 LER1	LD	A,22	;MEM FAULT
4E8F 01	00882	DB	1	;PROT A
4E90 3E14	00883 LER2	LD	A,20	;NOT A PRGM
4E92 B7	00884	OR	A	;SET ERR
4E93 C9	00885	RET		
	00886 ;			
4E94 FE02	00887 LXFR	CP	2	;XFR ADR?
4E96 2014	00888	JR	NZ,LLCOM	;NO
4E98 CDBB4E	00889	CALL	GB	;GET NXT BYT
4E9B FE02	00890	CP	2	;MUST BE 2
4E9D 20F1	00891	JR	NZ,LER2	;NO. A PRGM
4E9F CDBB4E	00892	CALL	GB	;GET XFR ADR

Listing Continued...

... Continued Listing

4EA2 6F	00893	LD	L,A	;STO LSB
4EA3 CDBB4E	00894	CALL	GB	;GET MSB
4EA6 67	00895	LD	H,A	;STO
4EA7 224B48	00896	LD	(W3),HL	;STO IN ADR
4EAA AF	00897	XOR	A	;NO ERR
4EAB C9	00898	RET		
	00899 ;			
4EAC FE20	00900 LLCOM	CP	32	;COMM?
4EAE 30E0	00901	JR	NC,LER2	;NOT A PRGM
4EB0 CDBB4E	00902 LCOM	CALL	GB	;GET COM LEN
4EB3 47	00903	LD	B,A	;XFR
4EB4 CDBB4E	00904 LCOMX	CALL	GB	;SKIP BYT
4EB7 10FB	00905	DJNZ	LCOMX	;LOOP
4EB9 18AE	00906	JR	LMAN	;CONT
	00907 ;			
4EBB D9	00908 GB	EXX		;SW REGS
4EBC 7D	00909	LD	A,L	;GET PTR LSB
4EBD B7	00910	OR	A	;GET SEC?
4EBE 2016	00911	JR	NZ,GB1	;NO
4EC0 FD4E00	00912	LD	C,(IY)	;GET DRV
4EC3 CD0946	00913	CALL	@RSEC	;READ SEC
4EC6 2802	00914	JR	Z,GB2	;OK
4EC8 D1	00915	POP	DE	;KILL INT CALL
4EC9 C9	00916	RET		
4ECA 1C	00917 GB2	INC	E	;BMP SEC
4ECB FD7E07	00918	LD	A,(IY+7)	;GET # SEC/TRK
4ECE E67F	00919	AND	127	;MASK
4ED0 BB	00920	CP	E	;BMP TRK?
4ED1 2003	00921	JR	NZ,GB1	;NO
4ED3 1E00	00922	LD	E,0	;RESET SEC
4ED5 14	00923	INC	D	;BMP TRK
4ED6 7E	00924 GB1	LD	A,(HL)	;GET BYT
4ED7 2C	00925	INC	L	;BMP PTR
4ED8 D9	00926	EXX		;SW REGS
4ED9 C9	00927	RET		
	00928 ;			
4EDA	00929 EXEC	EQU	\$	
4EDA E5	00930	PUSH	HL	;SAVE HL
4EDB CD534E	00931	CALL	LOAD	;LOAD FILE
4EDE E1	00932	POP	HL	;RESTO HL
4EDF C0	00933	RET	NZ	;ERR
4EE0 E5	00934	PUSH	HL	;SAVE HL
4EE1 2A4B48	00935	LD	HL,(W3)	;GET XFR ADR
4EE4 E3	00936	EX	(SP),HL	;RES HL, PUSH JMP
4EE5 C9	00937	RET		;XFR TO BIN
	00938 ;			
	00939 ;			
		ENQUE/DEQUE AN INTERRUPT RTN,		
		A=SLOT, DE->RTN		
	00940 ;			
4EE6 CDA04A	00941 ENQUE	CALL	SAVREG	;SAV REGS
4EE9 FE0A	00942	CP	10	;BAD SLOT?
4EEB D0	00943	RET	NC	;Y, RET
4EEC 210D47	00944	LD	HL,INTQUE	;QUE PTRS
4EEF 87	00945	ADD	A,A	;DOUBLE A

Listing Continued...

... Continued Listing

```

4EF0 85      00946      ADD      A,L      ;ADD PTR OFFSET
4EF1 6F      00947      LD        L,A      ;XFR
4EF2 F3      00948      DI          ;DISABLE INT
4EF3 73      00949      LD        (HL),E    ;STOR LSB RTN
4EF4 23      00950      INC       HL      ;BMP PTR
4EF5 72      00951      LD        (HL),D    ;STO MSB RTN
4EF6 FB      00952      EI          ;ENABLE INT
4EF7 C9      00953      RET
4EF8 116E49  00954  DEQUE  LD        DE,RETINT ;SLOT NULLER
4EFB 18E9    00955      JR        ENQUE      ;CONT
          00956      ;
4EFD        00957  ERRORD EQU      $
4EFD 32FA4D  00958      LD        (INT),A    ;STO VAL
4F00 211B4F  00959      LD        HL,@ERM1    ;PLAC TO ONV
4F03 4C      00960      LD        C,H      ;NO LEAD 0'S
4F04 CD0E4E  00961      CALL     CI2      ;CONV TO DEC
4F07 210D4F  00962      LD        HL,@ERM    ;GET MSG
4F0A C33646  00963      JP        @LINE    ;DISP, RET
4F0D        00964  @ERM  DM        'System error: '
          53 79 73 74 65 6D 20 65 72 72 6F 72
          3A 20
4F1B 30300D  00965  @ERM1 DM        '00',13
          00966      ;
4F1E        00967  DOSINI EQU      $
4F1E 215D4F  00968      LD        HL,KIDVR    ;NEW KI DVR
4F21 221640  00969      LD        (4016H),HL
4F24 213F3C  00970      LD        HL,3C3FH    ;VID
4F27 3E61    00971      LD        A,'a'      ;LC A
4F29 77      00972      LD        (HL),A    ;STO
4F2A BE      00973      CP        (HL)      ;LC MOD?
4F2B 3620    00974      LD        (HL),32    ;STO SPACE
4F2D 212750  00975      LD        HL,DODVR    ;VID DVR
4F30 2003    00976      JR        NZ,I001    ;NO LC MOD
4F32 221E40  00977      LD        (401EH),HL   ;STO NEW DVR
          00978      ;
4F35        00979  I001 EQU      $
4F35 21CA4D  00980      LD        HL,@INIT    ;INIT MSG
4F38 CD3646  00981      CALL     @LINE    ;DISP
4F3B 110002  00982  JPDOS LD        DE,200H   ;TRK2, SEC0, SYS
          ;INP
4F3E 0E00    00983      LD        C,0      ;DRIV 0
4F40 CD3046  00984      CALL     @LOAD    ;LOAD OBJ
4F43 2001    00985      JR        NZ,X12Z
4F45 E9      00986      JP        (HL)      ;JP TO PRGM
4F46 CD3346  00987  X12Z CALL     @ERROR    ;DIS ERR
4F49 CD4900  00988      CALL     49H      ;GET KEY
4F4C F3      00989      DI          ;KILL INT
4F4D C30D00  00990      JP        0DH      ;REBOOT
          00991      ;
          00992      ;      LOWER CASE/REPEAT DRIVERS
          00993      ;
4F50 00      00994  RPF   NOP
4F51 00      00995  LSTK  NOP
4F52 0000    00996  DELA  DW        0

```

Listing Continued ...

... Continued Listing

4F54 00	00997	RATE	NOP		;RATE
4F55 00	00998	CASE	NOP		
4F56	00999	KPB	DB	0,0,0,0,0,0,0	
		00 00 00 00 00 00 00			
	01000				
4F5D	01001	KIDVR	EQU	\$;KI DVR
4F5D CDE303	01002		CALL	3E3H	;SCAN KB
4F60 B7	01003		OR	A	;NEW KEY?
4F61 CAE94F	01004	R012	JP	Z,KI1	;N
4F64 210838	01005		LD	HL,3808H	;VID MEM
4F67 FE1A	01006		CP	26	;CTRL Z
4F69 2007	01007		JR	NZ,KC1	;N
4F6B CB56	01008		BIT	2,(HL)	;Z KEY ON?
4F6D 2003	01009		JR	NZ,KC1	;Y
4F6F AF	01010	KC0	XOR	A	;NUL SHIFT DWN
4F70 1877	01011		JR	KI1	;CON
4F72 FE1F	01012	KC1	CP	31	;CLEAR ASC 31?
4F74 2E80	01013		LD	L,80H	;HL=3880H, SHIFT
4F76 2004	01014		JR	NZ,KC2	;NOT CLEAR KEY
4F78 CB46	01015		BIT	0,(HL)	;SHIFT?
4F7A 28F3	01016		JR	Z,KC0	;N, NULL CLEAR
4F7C FE20	01017	KC2	CP	32	;SPACE?
4F7E 2013	01018		JR	NZ,KC3	;N
4F80 CB46	01019		BIT	0,(HL)	;SHIFT?
4F82 280F	01020		JR	Z,KC3	;N
4F84 2E10	01021		LD	L,10H	;TO 0 KEY
4F86 CB46	01022		BIT	0,(HL)	;ON?
4F88 2809	01023		JR	Z,KC3	;N
4F8A 3A554F	01024	R009	LD	A,(CASE)	;GET CASE FLAG
4F8D 2F	01025		CPL		;SWITCH
4F8E 32554F	01026	R010	LD	(CASE),A	;STO
4F91 18DC	01027		JR	KC0	;NUL SPACE
4F93 4F	01028	KC3	LD	C,A	;STO INP KEY
4F94 3A554F	01029	R011	LD	A,(CASE)	;GET CASE
4F97 B7	01030		OR	A	;NORM?
4F98 79	01031		LD	A,C	;GET CHR
4F99 281C	01032		JR	Z,KI6	;GO IF NORM
4F9B FE41	01033		CP	'A'	;LET?
4F9D 3818	01034		JR	C,KI6	;N
4F9F FE7B	01035		CP	'z'+1	;LET?
4FA1 300C	01036		JR	NC,KC4	;N
4FA3 FE61	01037		CP	'a'	;L/C?
4FA5 3004	01038		JR	NC,KC5	;SWITCH
4FA7 FE5B	01039		CP	'z'+1	;LET?
4FA9 300C	01040		JR	NC,KI6	;N
4FAB EE20	01041	KC5	XOR	20H	;SWITCH CASE
4FAD 1808	01042		JR	KI6	;CON
4FAF FE61	01043	KC4	CP	'a'	;L/C?
4FB1 3804	01044		JR	C,KI6	;N
4FB3 FE7B	01045		CP	'z'+1	;L/C?
4FB5 30F4	01046		JR	NC,KC5	;N
4FB7 FE1F	01047	KI6	CP	31	;CLR?
4FB9 2810	01048		JR	Z,KI6A	;Y
4FBB 2E40	01049		LD	L,40H	;HL=3840H

Listing Continued...

... Continued Listing

4FBD CB4E	01050	BIT	1, (HL)	;CLEAR PRESSED?
4FBF 280A	01051	JR	Z, KI6A	;N
4FC1 FE60	01052	CP	' '	;L/C?
4FC3 30E6	01053	JR	NC, KC5	;MAKE U/C
4FC5 E67F	01054	AND	7FH	;GET REL FUNC
				;CODE
4FC7 D620	01055	SUB,	32	;MAK REL
4FC9 C680	01056	ADD	A, 128	;MAKE FUNCTION
				;CODE
4FCB	01057 R004	EQU	\$	
4FCB 32514F	01058 KI6A	LD	(LSTK), A	;STO KEY
4FCE F5	01059	PUSH	AF	;SAV CHR
4FCF AF	01060	XOR	A	;CLR RPT DELAY
4FD0 32504F	01061 R001	LD	(RPF), A	;RPT OFF
4FD3 218001	01062	LD	HL, 180H	;DELAY TIME
4FD6 22524F	01063 R005	LD	(DELA), HL	;INIT
4FD9 213640	01064	LD	HL, 4036H	;KEYS PRESSED
				;BUFFER
4FDC 11564F	01065 R013	LD	DE, KPB	;RPT BUF
4FDF 010700	01066	LD	BC, 7	;# BYTS
4FE2 EDB0	01067	LDIR		;MOVE
4FE4 F1	01068	POP	AF	;GET CHR
4FE5 C9	01069	RET		;RET
	01070 ;			
4FE6 AF	01071 KI5	XOR	A	;NO KEY
4FE7 18E2	01072	JR	KI6A	;CON
4FE9 213640	01073 KI1	LD	HL, 4036H	;KEY BUFF
4FEC 11564F	01074 R014	LD	DE, KPB	;RPT BUF
4FEF 0607	01075	LD	B, 7	;CHRS TO TEST
4FF1 1A	01076 KI2	LD	A, (DE)	;GET KEY BYT
4FF2 BE	01077	CP	(HL)	;SAME?
4FF3 20F1	01078	JR	NZ, KI5	;N
4FF5 23	01079	INC	HL	;BMP PTRS
4FF6 13	01080	INC	DE	
4FF7 10F8	01081	DJNZ	KI2	;LOOP
4FF9 3A504F	01082 R002	LD	A, (RPF)	;RPT ON?
4FFC B7	01083	OR	A	;?
4FFD 2017	01084	JR	NZ, KI4	;Y
4FFF 2A524F	01085 R006	LD	HL, (DELA)	;DELAY CNT
5002 2B	01086	DEC	HL	;START RPT?
5003 22524F	01087 R007	LD	(DELA), HL	;RES
5006 7D	01088	LD	A, L	
5007 B4	01089	OR	H	;START RPT?
5008 2802	01090	JR	Z, KI3	;Y
500A AF	01091	XOR	A	;NO KEY
500B C9	01092	RET		
500C 3E46	01093 KI3	LD	A, 70	;RATE VAL
500E 32544F	01094 R008	LD	(RATE), A	;STO
5011 32504F	01095 R003	LD	(RPF), A	;RPT ON
5014 AF	01096	XOR	A	;NO KEY
5015 C9	01097	RET		
5016	01098 R015	EQU	\$	
5016 21544F	01099 KI4	LD	HL, RATE	;RATE BUF;RATE
				;BUF

Listing Continued...

... Continued Listing

5019 35	01100	DEC	(HL)	;GIVE RPT KEY?
501A 2802	01101	JR	Z,KB1	;Y
501C AF	01102	XOR	A	;NO KEY
501D C9	01103	RET		
501E 3E46	01104 KB1	LD	A,70	;RATE
5020 32544F	01105 R016	LD ,	(RATE),A	;RES RATE
5023	01106 R017	EQU	\$	
5023 3A514F	01107	LD	A,(LSTK)	;GET KEY VAL
5026 C9	01108	RET		
	01109 ;			
	01110 ;			
	01111 ;			
		VIDEO DRIVER		
5027	01112 DODVR	EQU	\$	
5027 3812	01113	JR	C,DD1	;RET CUR CHAR
5029 79	01114	LD	A,C	;GET CHR
502A FE41	01115	CP	'A'	;LET?
502C 380C	01116	JR	C,DD0	;N
502E FE7B	01117	CP	'z'+1	;GARF?
5030 3008	01118	JR	NC,DD0	;Y
5032 FE61	01119	CP	'a'	;LC CHR?
5034 3008	01120	JR	NC,DD2	;Y
5036 FE5B	01121	CP	'z'+1	;U/C?
5038 3804	01122	JR	C,DD2	;Y
503A B7	01123 DD0	OR	A	;NC
503B C35804	01124 DD1	JP	458H	;NORM DISP ENTRY
503E DD6E03	01125 DD2	LD	L,(IX+3)	
5041 DD6604	01126	LD	H,(IX+4)	;GET CURS ADR
5044 DD7E05	01127	LD	A,(IX+5)	;GET CURS CHR
5047 B7	01128	OR	A	;CUR ON?
5048 2801	01129	JR	Z,DD3	;N
504A 77	01130	LD	(HL),A	;DISP O/L CHR
504B 79	01131 DD3	LD	A,C	;GET CHR TO DISP
504C C37D04	01132	JP	47DH	;DISP CHR AS IS
	01133 ;			
	01134 ;			
	01135 ;			
		PRINTER DRIVER EXTENTSION		
504F 3A2940	01136 PRDVR	LD	A,(4029H)	;LINE COUNT
5052 47	01137	LD	B,A	;STO
5053 3A2840	01138	LD	A,(4028H)	;MAX COUNT
5056 90	01139	SUB	B	;SUB COUNT
5057 FE06	01140	CP	6	;TIME TO PAGE?
5059 2011	01141	JR	NZ,Z0	;N
505B C5	01142	PUSH	BC	;SAVE C=CHR
505C 0606	01143	LD	B,6	;# OF LINES
505E 0E0A	01144	LD	C,10	;LINE FEED
5060 C5	01145 C2	PUSH	BC	;SAVE REGS
5061 CD8D05	01146 CALL1	CALL	58DH	;CALL DVR
5064 C1	01147	POP	BC	;GET VALS
5065 10F9	01148	DJNZ	C2	;LOOP
5067 AF	01149	XOR	A	;CLR LINE COUNT
5068 322940	01150	LD	(4029H),A	;RESET
506B C1	01151	POP	BC	;GET CHR TO PR
506C	01152 Z0	EQU	\$	
506C 79	01153	LD	A,C	;GET CHR

Listing Continued...

... Continued Listing

506D	FE0A	01154	CP	10	
506F	2004	01155	JR	NZ,ZX	;LINE FEED?
5071	2A2940	01156	LD	HL,(4029H)	;N
5074	34	01157	INC	(HL)	;COUNTER
5075		01158	ZX	EQU	;BMP LINE COUNT
5075	C38D05	01159	CALL2	JP	
		01160	;	58DH	;JP TO DVR
5078		01161	EOP	EQU	
5078		01162	END	DOSINI	

```

*****
00001
*
00002 ;**      S/OS - SMALL/OPERATING SYSTEM      *
*
00003 ;**      USER INTERFACE MODULE              *
*
00004 ;**      VERSION 1.0 - MODEL I                *
*
00005 ;**      CREATED: 08/05/82                    *
*
00006 ;**      UPDATED: 08/06/82                    *
*
00007 ;**      (C) 1982 by Michael Wagner           *
*
00008
*****
00009 ;
00010 ;      Label equates
00011 ;
4630 00012 @LOAD EQU      4630H      ;LOAD OJB CODE
4636 00013 @LINE EQU      4636H      ;LINE VIDEO
4639 00014 @PRINT EQU     4639H
4633 00015 @ERROR EQU     4633H      ;ERROR DISP
4800 00016 INPBUF EQU     4800H      ;DOS USER INPUT
                                         ;BUFFER
4840 00017 DRIVES EQU     4840H      ;HIGHEST DRIVE #
00018 ;
5100 00019 ORG      5100H
00020 ;
5100 0000 00021 W1      DW      0      ;WORK REGS
5102 0000 00022 W2      DW      0
5104 0000 00023 W3      DW      0
5106 00024 @MSG      DM      10,'S/OS',13
                        0A 53 2F 4F 53 0D
00025 ;
510C 00026 START      EQU      $
510C FB 00027 EI
510D 31FC41 00028 LD      SP,41FCH      ;INT ON
5110 3A2040 00029 LD      A,(4020H)      ;INIT STACK
5113 E63F 00030 AND      63      ;GET CURSOR LOC
5115 2805 00031 JR      Z,S00      ;GET LINE POSN
5117 3E0D 00032 LD      A,13      ;AT LINE STA
5119 CD3300 00033 CALL    33H      ;WRITE A C/R
                                         ; TO THE VIDEO
511C 210651 00034 S00    LD      HL,@MSG      ;GET 'S/OS' MSG
511F CD3646 00035 CALL    @LINE      ;DISP
5122 3E3F 00036 S01    LD      A,'?'      ;GET PROMPT
5124 CD3300 00037 CALL    33H      ;WRITE TO THE
                                         ;VIDEO
5127 210048 00038 LD      HL,INPBUF      ;DOS INP BUF
512A 063F 00039 LD      B,63      ;MAX IMP
512C CD4000 00040 CALL    40H      ;GET INP
512F 38F1 00041 JR      C,S01      ;BREAK HIT
5131 01B851 00042 LD      BC,@ERR1      ;'DRIVE'
5134 CD4852 00043 CALL    NEXT      ;GET NEXT CHR
5137 28E9 00044 JR      Z,S01      ;NULL LINE, REDO

```

Listing Continued...

... Continued Listing

5139	CDD751	00045	CALL	GETINP	;GET DRIVE #
513C	384F	00046	JR	C,DISE	;GO IF OVERFLOW
513E	204D	00047	JR	NZ,DISE	;GO IF MSB >0
5140	3A4048	00048	LD	A,(DRIVES)	;GET # DRIVES IN
					;SYS
5143	BB	00049	CP	E	;TOO BIG?
5144	3847	00050	JR	C,DISE	;YES
5146	7B	00051	LD	A,E	;GET DRIVE #
5147	320051	00052	LD	(W1),A	;STO DRIVE #
514A	01BE51	00053	LD	BC,@ERR2	; 'TRACK' MSG
514D	CD4852	00054	CALL	NEXT	; POSN TO TRKSPEC
5150	283B	00055	JR	Z,DISE	;NO TRACK
5152	CDD751	00056	CALL	GETINP	;GET TRK #
5155	3836	00057	JR	C,DISE	;OVERFLOW ERR
5157	2034	00058	JR	NZ,DISE	;GO IF D>0
5159	320251	00059	LD	(W2),A	;STO TRK
515C	01C451	00060	LD	BC,@ERR3	; 'SECTOR' MSG
515F	CD4852	00061	CALL	NEXT	; POSN TO SEC
5162	2829	00062	JR	Z,DISE	;NO SEC SPEC
5164	CDD751	00063	CALL	GETINP	;GET SEC #
5167	3824	00064	JR	C,DISE	;OVERFLOW
5169	2022	00065	JR	NZ,DISE	;D>0 ERR
516B	CD4852	00066	CALL	NEXT	; POSN HL TO
					;PARAMS
516E	220451	00067	LD	(W3),HL	;STO PARAM PTR
5171	3A0251	00068	LD	A,(W2)	;GET TRK
5174	57	00069	LD	D,A	;STO TRACK
5175	3A0051	00070	LD	A,(W1)	;GET DRIVE
5178	4F	00071	LD	C,A	;XFR TO DRIVE REG
5179	CD3046	00072	CALL	@LOAD	;LOAD OBJ CODE
517C	2009	00073	JR	NZ,ERROR	;GO IF ERROR
517E	112D40	00074	LD	DE,402DH	;DOS RET
5181	D5	00075	PUSH	DE	;FOR RET ADR
5182	E5	00076	PUSH	HL	;PUSH XFER ADR
5183	2A0451	00077	LD	HL,(W3)	;GET PARAM PTR
5186	C9	00078	RET		;JMP TO RTN
		00079			
5187	CD3346	00080	CALL	@ERROR	;DISP SYS ERROR
518A	C30C51	00081	JP	START	;PROMPT
		00082			
518D	C5	00083	DISE	PUSH	BC
518E	21A051	00084	LD	HL,@ERR	;SAVE UNIT MSG
5191	CD3646	00085	CALL	@LINE	;ERR MSG
5194	E1	00086	POP	HL	;DISPLAY
5195	CD3646	00087	CALL	@LINE	;GET UNIT MSG
5198	21CB51	00088	LD	HL,@ERR4	;DISP
519B	CD3646	00089	CALL	@LINE	;REMAINING MSG
519E	18EA	00090	JR	STARTV	;DISP
		00091			;RESTART
51A0		00092	@ERR	DM	'* * Invalid or missing ',3
			2A 20	2A 20 49	6E 76 61 6C 69 64 20
			6F 72	20 6D 69	73 73 69 6E 67 20 03
51B8		00093	@ERR1	DM	'drive',3
			64 72	69 76 65	03

Listing Continued...

... Continued Listing

```

51BE      00094 @ERR2    DM      'track',3
          74 72 61 63 6B 03
51C4      00095 @ERR3    DM      'sector',3
          73 65 63 74 6F 72 03
51CB      00096 @ERR4    DM      ' number * *',13
          20 6E 75 6D 62,65 72 20 2A 20 2A 0D
          00097 ;
          00098 ;
          00099 ;      GETINP2 08/04/82
          00100 ;      CONVERT DEC TO TWO BYTE INT
          00101 ;      RANGE 0-65535 (0-FFFFH)
          00102 ;      ENTRY:  HL-> DEC NUMBER
          00103 ;      EXIT:   DE=DEC VALUE
          00104 ;      Z= MSB=0 (D=0)
          00105 ;      CF= OVERFLOW
          00106 ;      HL->TERMINATING CHR
          00107 ;
51D7      00108 GETINP   EQU      $
51D7 E5    00109         PUSH     HL              ;SAVE LINPTR
51D8 110000 00110         LD      DE,0           ;ZERO ACCUM
51DB 7E     00111 GI0    LD      A,(HL)          ;GET CHR
51DC FE48   00112         CP      'H'           ;HEX NUM?
51DE 2831   00113         JR      Z,HEXINP       ;NUM IS HEX
51E0 D630   00114         SUB     '0'           ;MAKE CHR BIN
51E2 3827   00115         JR      C,DDONE        ;TERM HIT
51E4 FE0A   00116         CP      10            ;BAD RANGE?
51E6 3029   00117         JR      NC,HEXINP      ;TRY HEX INP
51E8 23     00118         INC     HL             ;BMP LINPTR
51E9 E5     00119         PUSH     HL            ;SAVE PTR
51EA 62     00120         LD      H,D           ;HL=ACCUM
51EB 6B     00121         LD      L,E
51EC E5     00122         PUSH     HL            ;SAVE REGS
51ED D5     00123         PUSH     DE
51EE 119A19 00124         LD      DE,6554        ;ACCUM MUST LESS
51F1 B7     00125         OR      A             ;NC SET
51F2 ED52   00126         SBC     HL,DE          ;SUB 6553 FROM
          ;ACCUM
51F4 D1     00127         POP      DE            ;RESTO REGS
51F5 E1     00128         POP      HL
51F6 300E   00129         JR      NC,G01         ;RESULT WILL
          ;OVERFLOW
51F8 29     00130         ADD     HL,HL          ;TIMES BY 10
51F9 29     00131         ADD     HL,HL
51FA 19     00132         ADD     HL,DE
51FB 29     00133         ADD     HL,HL
51FC 1600   00134         LD      D,0           ;ZERO D
51FE 5F     00135         LD      E,A           ;DE=NEW DIGIT TO
          ;ADD
51FF 19     00136         ADD     HL,DE          ;ADD NEW DIGIT
5200 3804   00137         JR      C,G01         ;GO IF OVERFLOW
5202 EB     00138         EX      DE,HL         ;PASS ACCUM TO DE
5203 E1     00139         POP      HL            ;GET LINPTR
5204 18D5   00140         JR      GI0           ;DO NEXT CHR
          00141 ;

```

Listing Continued...

... Continued Listing

5206 E3	00142 G01	EX	(SP),HL	;KILL OLD LINPTR
5207 E1	00143	POP	HL	;KEEP CUR PTR
5208 37	00144	SCF		;SET OVERFLOW
				;FLAG
5209 1803	00145	JR	DDONE1	;CONT
520B 7A	00146 DDONE	LD	A,D	;GET MSB OF
				;RESULT
520C B7	00147	OR	A	;SET NZ,Z
520D 7B	00148	LD	A,E	;A=LSB VALUE
520E E3	00149 DDONE1	EX	(SP),HL	;SAVE CUR LINPTR
	00150			;HL=ORG LINPTR
520F E1	00151	POP	HL	;HL=CUR LINPTR
5210 C9	00152	RET		
	00153 ;			
5211 E1	00154 HEXINP	POP	HL	;RESTOR LINPTR
5212 110000	00155	LD	DE,0	;RESET ACCUM
5215 7E	00156 H01	LD	A,(HL)	;GET NEXT CHR
5216 FE48	00157	CP	'H'	;END OF NUM?
5218 2826	00158	JR	Z,HDONEX	;YES
521A D630	00159	SUB	'0'	;DONE?
521C 3F	00160	CCF		;?
521D 3022	00161	JR	NC,HDONE	;Y
521F FE17	00162	CP	'G'-'0'	;DONE?
5221 301E	00163	JR	NC,HDONE	;Y
5223 FE0A	00164	CP	10	;DEC DIGIT?
5225 3807	00165	JR	C,H02	;Y
5227 FE11	00166	CP	17	;HEX DIGIT?
5229 3F	00167	CCF		;SWI CF
522A 3015	00168	JR	NC,HDONE	;N, TERM
522C D607	00169	SUB	7	;MAKE A-F REL
522E 23	00170 H02	INC	HL	;BMP LINPTR
522F F5	00171	PUSH	AF	;SAVE CUR DIGIT
5230 7A	00172	LD	A,D	;WILL NEW CHR
				;OVERFLOW LINE?
				;?
5231 E6F0	00173	AND	0F0H	;YES
5233 2010	00174	JR	NZ,H03	;GET NEW CHR
5235 F1	00175	POP	AF	;HL=ACCUM
5236 EB	00176	EX	DE,HL	;SHIFT ACCUM 4<-
5237 29	00177	ADD	HL,HL	
5238 29	00178	ADD	HL,HL	
5239 29	00179	ADD	HL,HL	
523A 29	00180	ADD	HL,HL	
523B B5	00181	OR	L	;CMP NEW LSB
523C 6F	00182	LD	L,A	;XFER TO ACCUM
523D EB	00183	EX	DE,HL	;DE=ACCUM
523E 18D5	00184	JR	H01	;DO NEXT CHR
	00185 ;			
5240 23	00186 HDONEX	INC	HL	;BMP LINPTR
5241 7A	00187 HDONE	LD	A,D	;GET MSB INP
5242 B7	00188	OR	A	;SET NZ,Z
5243 7B	00189	LD	A,E	;A=LSB VALUE
5244 C9	00190	RET		
5245 F1	00191 H03	POP	AF	;GET LAST CHR
5246 37	00192	SCF		;SET OVERFLOW ERR

Listing Continued...

... Continued Listing

5247 C9	00193	RET		
	00194 ;			
5248 7E	00195 NEXT	LD	A, (HL)	;GET CHR
5249 FE0D	00196	CP	13	;C/R?
524B C8	00197	RET	Z	;END OF LINE
524C FE2C	00198	CP	' , '	;COMMA?
524E 2803	00199	JR	Z, NEXT1	;YES, IGNORE
5250 FE20	00200	CP	32	;SPACE?
5252 C0	00201	RET	NZ	;NO, RET
5253 23	00202 NEXT1	INC	HL	;BMP PTR
5254 18F2	00203	JR	NEXT	;LOOP
	00204 ;			
5256	00205	END	START	
00000	total errors			

notes

Appendix I

The Best Term Program in This Book

As an extra bonus, I threw in this terminal program that we used in preparing this book. It is somewhat of an intelligent terminal. It allows instant resetting of the RS-232 parameters, buffering of incoming data, disk file I/O to the buffer, and various other functions. TERM will assemble in any TRS-80 assembler without problems.

TERM supports the baud rates: 110, 150, 300, 600, 1200, 2400, 9600; the word lengths: 5, 6, 7 and 8; the number of stop bits: 1 and 2; the parity check: Even, Odd and None.

Running TERM

Since TERM is a machine-language program, it will be entered from DOS. When TERM is entered, it will announce itself, then immediately prompt you for the RS-232 settings (baud, parity, etc). You will then find yourself in the TERM menu. Most functions are selected from here. Here is what the TERM menu looks like:

- B – Assign the BREAK key value
- C – Clear contents of buffer
- D – Display contents of buffer
- E – Toggle “echo transmit data” switch
- L – Load a file into the buffer
- P – Print contents of buffer
- Q – Display status of RS-232 and buffer
- R – Set RS-232 parameters
- S – Transmit buffer data
- T – Enter terminal mode
- W – Write buffer to a file
- X – Exit to DOS

Explanation of the Menu Functions

B – Assign the break key value

This one is pretty self-explanatory. It lets you assign the break key a particular value. Most host computers use CONTROL:C (ASCII 3) for the break character.

C – Clear the contents of the buffer

TERM has the capability of buffering incoming data, and/or load disk files into the buffer for transmission. This command simply clears the buffer of any data.

D – Display contents of the buffer

This command displays the contents of the buffer. you may pause by pressing (SHIFT) '@'. You may abort the display by pressing break. The buffer is displayed from start to finish (unless break is hit).

E – Toggle 'echo transmit data' switch

When in terminal mode (see T command), echo displays what you are sending. This is for systems that do not echo your characters back to you. If you are communicating with a system that echos your character and you have echo ON, you will see double characters when you send. This echo does NOT work when using the SEND (command S) command.

L – Load a file into the buffer

This loads a disk file into TERM's buffer. If the buffer already contains data, the new file will be concatenated to the end of the current buffer contents. This allows multiple file input. The filename must follow the L command, e.i: L NAME/TX.

P – Print (line-print) contents of buffer

This command line-prints the contents of the buffer. you may pause by pressing (SHIFT) '@'. You may abort the printing by pressing break. The buffer is printed from start to finish (unless break is hit).

Q – Display RS-232 and buffer status

This displays the current RS-232 settings, how many characters in the buffer, and how many unused bytes remain in the buffer.

R – Set the RS-232 parameters

This invokes the same protocol queries that you answered at the start of the program. This allows you to re-program the RS-232 whenever you desire.

S – Send buffer contents

This command transmits the contents of the buffer to the host computer. You may pause by pressing (SHIFT) '@'. You may abort the send by pressing break. The buffer is sent from start to finish (unless break is hit).

T – Enter terminal mode

When this function is selected, you will enter the terminal mode. This lets you directly communicate via the keyboard and video monitor to the host computer. When in the terminal mode, the CLEAR key becomes your CONTROL key, e.g. CLEAR: C sends a control C. You may buffer incoming data at any time by pressing CLEAR Q. You may turn off the buffering by pressing CLEAR P. You may exit the terminal back to the menu by pressing SHIFT BREAK. The incoming data buffering is always OFF when entering the terminal mode.

W – Write buffer to a disk file

This allows you to write the buffer to a disk file for loading into your word processor or whatever. The filename must follow the W, e.i: W STOCK/PCL.

X – Exit to DOS

This simply exits to DOS via the **402D** DOS vector.

```

00001 ;      TERM - SMART TERMINAL PROGRAM
00002 ;      VERSION 1.2 - 09/09/82
00003 ;
5200      00004      ORG      5200H
00005 ;
5200      00006 BUFFER DEFS      256      ; I/O BUFF
5300      00007 FCB      DEFS'    32      ; FCB
5320 00      00008 ACURS      NOP      ; CURR RS-232 PRGM
5321 00      00009 ABAUDV      NOP      ; CUR BAUD VAL
00010 ;
00011 ;      RS232 - I/O DRIVERS
00012 ;
00E8      00013 MSTAT      EQU      0E8H      ; MASTER RESET/MODEM STAT
00E9      00014 BAUD      EQU      0E9H      ; BAUD SELECT/SENSE SW
00EA      00015 CTRL      EQU      0EAH      ; CONTROL AND RS232 STAT
00EB      00016 DATA      EQU      0EBH      ; DATA I/O
0040      00017 INITB      EQU      040H      ; INIT BYTE
5322 00      00018 VIDF      NOP      ; VIDEO FILTER FLAG
00019 ;
00020 ;      GETS A BYTES FROM THE UART/NO WAIT
00021 ;
5323      00022 INPUT      EQU      $
5323 F3      00023      DI
5324 DBEA      00024      IN      A, (CTRL)      ; DISABLE INTERRUPTS
5326 07      00025      RLCA      ; GET STAT REG
5327 3028      00026      JR      NC, NODATA      ; GET DRQ IN CARRY
5329 DBEB      00027      IN      A, (DATA)      ; GO IF NO DATA RCVD
532B 4F      00028      LD      C, A      ; GET BYTE
532C 3AED55      00029      LD      A, (AHIB)      ; SAVE BYTE
532F B7      00030      OR      A      ; HI BIT STRIP?
5330 2802      00031      JR      Z, IBT      ; SET
5332 CBB9      00032      RES      7, C      ; N
5334 DBEA      00033 IBT      IN      A, (CTRL)      ; KILL HI BIT
5336 E638      00034      AND      38H      ; GET STAT
5338 280F      00035      JR      Z, OK      ; ERROR?
533A 0F      00036      RRCA      ; IF NOT
533B 0F      00037      RRCA
533C 0F      00038      RRCA
533D 0F      00039      RRCA
533E 0E9F      00040      LD      C, 159
5340 3807      00041      JR      C, OK
5342 0E97      00042      LD      C, 151      ; PARITY ERR
5344 0F      00043      RRCA
5345 3802      00044      JR      C, OK      ; FRAME ERR
5347 0EBF      00045      LD      C, 191      ; OVER RUN
5349 79      00046 OK      LD      A, C      ; GET CHAR
534A 00      00047      NOP
534B 00      00048      NOP
534C 00      00049      NOP
534D 00      00050      NOP
534E B7      00051 GEB      OR      A
534F FB      00052      EI      ; GET FLAGS
5350 C9      00053      RET      ; ENABLE INT
5351 37      00054 NODATA      SCF
5352 FB      00055      EI      ; NO CHR
; ENABLES INT

```

```

5353 C9      00056      RET                      ;RETURN
            00057 ;
            00058 ;      PUTS A BYTE TO THE UART
            00059 ;
5354      00060 OUTPUT EQU      $
5354 F3      00061      DI                      ;DISABLE INT
5355 4F      00062      LD      C,A              ;SAVE BYTE
5356 DBEA    00063 OUTI      IN      A,(CTRL)      ;GET STATUS
5358 CB77    00064      BIT      6,A              ;CHECK TRANSMIT HOLDING
535A 28FA    00065      JR      Z,OUTI            ;GO IF CURRENT BYTE NOT
            00066 ;                              ;GONE YET
535C 79      00067      LD      A,C              ;GET BYTE
535D D3EB    00068      OUT      (DATA),A          ;OUTPUT BYTE
535F FB      00069      EI                      ;ENABLE INT
5360 C9      00070      RET                      ;RETURN
            00071 ;
            00072 ;      INITIALIZED RS-232 UART
            00073 ;
5361      00074 INITRS EQU      $
5361 D3E8    00075      OUT      (MSTAT),A          ;RESETS RS-232
5363 D3E9    00076      OUT      (BAUD),A          ;SELECT BAUD
5365 79      00077      LD      A,C              ;GET CTRL WORD
5366 D3EA    00078      OUT      (CTRL),A          ;LOAD INFO
5368 C9      00079      RET                      ;RETURN
            00080 ;
            00081 ;      TAKES NEXT DECIMAL INPUT AT (HL) AND PUTS IT IN
            00082 ;      DE. C SET IS NUMBER WAS BIGGER THAN 65530
            00083 ;      ALL REGS USE. HL=CHAR AFTER LAST NUMBER
            00084 ;
5369      00085 NEXT EQU      $
5369 7E      00086      LD      A,(HL)              ;GET CHR
536A FE0D    00087      CP      13                ;C/R?
536C C8      00088      RET      Z                ;YES. EOL
536D FE20    00089      CP      32                ;SPACE?
536F C0      00090      RET      NZ               ;NO
5370 23      00091      INC      HL                ;BMP
5371 18F6    00092      JR      NEXT              ;LOOP
            00093 ;
            00094 ;      PROGRAM START
            00095 ;
5373      00096 START EQU      $
5373 DBEB    00097      IN      A,(DATA)            ;INP DATA REG
5375 FEFF    00098      CP      255                ;IS RS232 THERE?
5377 C2CA53  00099      JP      NZ,SOK            ;YES
537A 218253  00100      LD      HL,RSN            ;ERR MSG
537D CD045C  00101      CALL    DLINE            ;DIS
5380 AF      00102      XOR      A                ;NO ER
5381 C9      00103      RET
5382      00104 RSN      DEFB      'System not equipped with RS232!'
                    53 79 73 74 65 6D 20 6E 6F 74 20 65
                    71 75 69 70 70 65 64 20 77 69 74 68
                    20 52 53 32 33 32 21
53A1 0D      00105      DEFB      13
53A2 0F      00106 INITM DEFB      15

```

Term Program

53A3	00107	DEFM	'TERM - MJW'	
		54 45 52 4D 20	2D 20 4D 4A 57	
53AD 27	00108	DEFB	39	
53AE	00109	DEFM	'S TERMINAL PROGRAM VER 1.1'	
		53 20 54 45 52	4D 49 4E 41 4C 20 50	
		52 4F 47 52 41	4D 20 56,45 52 20 31	
		2E 31		
53C8 0A	00110	DEFB	10	
53C9 0D	00111	DEFB	13	
53CA	00112	SOK EQU	\$	
53CA CDC901	00113	CALL	1C9H	;CLS
53CD 21A253	00114	LD	HL,INITM	;INIT MSG
53D0 CD045C	00115	CALL	DLINE	;DIS
53D3 2A2040	00116	LD	HL,(4020H)	;GET CUR LOC
53D6 22185A	00117	LD	(CURLO),HL	
53D9 CDAD5A	00118	CALL	RESET	;SET RS-232 PARAMS
53DC	00119	MAIN EQU	\$	
53DC 215F54	00120	LD	HL,MENUD	;GET MENU
53DF CD045C	00121	CALL	DLINE	;DIS
53E2 2A2040	00122	LD	HL,(4020H)	;GET CURLOC
53E5 22185A	00123	LD	(CURLO),HL	;STOR
53E8 31FC41	00124	INCMD LD	SP,41FCH	;REINIT
53EB 21E853	00125	LD	HL,INCMD	;FOR RET
53EE E5	00126	PUSH	HL	
53EF 2A185A	00127	LD	HL,(CURLO)	;PLACE TO PUT
53F2 222040	00128	LD	(4020H),HL	;CUR LOC
53F5 3E1E	00129	LD	A,30	;ERASE TO EOL
53F7 CD3300	00130	CALL	33H	;DIS
53FA AF	00131	XOR	A	;RESET
53FB 32AD58	00132	LD	(SPF),A	; MEM SPOOL FLAG
53FE 3E3E	00133	LD	A,'>'	;PROMPT
5400 CD3300	00134	CALL	33H	;DIS
5403 211843	00135	LD	HL,4318H	;IN BUFF
5406 0620	00136	LD	B,32	;MAX INP
5408 CDBC59	00137	CALL	LIR	;GET INP
540B 38CF	00138	JR	C,MAIN	;BREAK, DIS MENU
540D 3E1F	00139	LD	A,31	;CAA
540F CD3300	00140	CALL	33H	;CLR TO END OF SCR
5412 CD6953	00141	CALL	NEXT	;POSN TO CHR
5415 C8	00142	RET	Z	;ENTER HIT
	00143 ;			
	00144 ;			
	00145 ;			
		BRANCHES **		
5416 CBAF	00146	RES	5,A	;FORCE LOWCASE
5418 FE42	00147	CP	'B'	;BREAK KEY VAL?
541A CA2056	00148	JP	Z,DEFBRK	
541D FE58	00149	CP	'X'	;DOS?
541F CA2D40	00150	JP	Z,402DH	
5422 FE50	00151	CP	'P'	;PRINT BUFF?
5424 CADB58	00152	JP	Z,PRINT	;GO
5427 FE54	00153	CP	'T'	;TERM MODE?
5429 CA5A58	00154	JP	Z,TERM	
542C FE51	00155	CP	'Q'	;QUERY STAT?
542E CA9A56	00156	JP	Z,BSTAT	
5431 FE45	00157	CP	'E'	;MODE?

```

5433 CA1557 00158 JP Z,ECHO
5436 FE52 00159 CP 'R' ;RESET?
5438 CAAD5A 00160 JP Z,RESET
543B FE57 00161 CP 'W' ;SAVE BUFF?
543D CADE58 00162 JP Z,SAVE
5440 FE53 00163 CP 'S' ;SEND BUFF?
5442 CAD358 00164 JP Z,SEND
5445 FE44 00165 CP 'D' ;DISPLAY BUFF?
5447 CAD858 00166 JP Z,DISP
544A FE4C 00167 CP 'L' ;LOAD BUFF?
544C CA7A59 00168 JP Z,LOAD
544F FE41 00169 CP 'A' ;SEN ASCII?
5451 CAE95B 00170 JP Z,ASCII
5454 FE48 00171 CP 'H' ;TOG HI BIT STRIP
5456 CA0256 00172 JP Z,BITS ;Y
5459 FE43 00173 CP 'C' ;CLEAR BUFF?
545B CA4457 00174 JP Z,CLEAR
545E C9 00175 RET ;NO CMD, MENU
545F 00176 MENUD EQU $
545F 0F 00177 DEFB 15
5460 1C 00178 DEFB 28
5461 1F 00179 DEFB 31
5462 00180 DEFM 'B - Assign the BREAK key value'
42 20 2D 20 41 73 73 69 67 6E 20 74
68 65 20 42 52 45 41 4B 20 6B 65 79
20 76 61 6C 75 65
5480 0A 00181 DEFB 10
5481 00182 DEFM 'C - Clear contents of buffer'
43 20 2D 20 43 6C 65 61 72 20 63 6F
6E 74 65 6E 74 73 20 6F 66 20 62 75
66 66 65 72
549D 0A 00183 DEFB 10
549E 00184 DEFM 'D - Display contents of buffer'
44 20 2D 20 44 69 73 70 6C 61 79 20
63 6F 6E 74 65 6E 74 73 20 6F 66 20
62 75 66 66 65 72
54BC 0A 00185 DEFB 10
54BD 00186 DEFM 'E - Toggle "echo transmit data" switch'
45 20 2D 20 54 6F 67 67 6C 65 20 22
65 63 68 6F 20 74 72 61 6E 73 6D 69
74 20 64 61 74 61 22 20 73 77 69 74
63 68
54E3 0A 00187 DEFB 10
54E4 00188 DEFM 'L - Load a file into the buffer'
4C 20 2D 20 4C 6F 61 64 20 61 20 66
69 6C 65 20 69 6E 74 6F 20 74 68 65
20 62 75 66 66 65 72
5503 0A 00189 DEFB 10
5504 00190 DEFM 'P - Print contents of buffer'
50 20 2D 20 50 72 69 6E 74 20 63 6F
6E 74 65 6E 74 73 20 6F 66 20 62 75
66 66 65 72
5520 0A 00191 DEFB 10
5521 00192 DEFM 'Q - Display status of RS-232 and buffer'

```

```

51 20 2D 20 44 69 73 70 6C 61 79 20
73 74 61 74 75 73 20 6F 66 20 52 53
2D 32 33 32 20 61 6E 64 20 62 75 66
66 65 72
5548 0A      00193      DEFB      10
5549          00194      DEFM      'R - Set, RS-232 parameters'
52 20 2D 20 53 65 74 20 52 53 2D 32
33 32 20 70 61 72 61 6D 65 74 65 72
73
5562 0A      00195      DEFB      10
5563          00196      DEFM      'S - Transmit buffer data'
53 20 2D 20 54 72 61 6E 73 6D 69 74
20 62 75 66 66 65 72 20 64 61 74 61
557B 0A      00197      DEFB      10
557C          00198      DEFM      'T - Enter terminal mode'
54 20 2D 20 45 6E 74 65 72 20 74 65
72 6D 69 6E 61 6C 20 6D 6F 64 65
5593 0A      00199      DEFB      10
5594          00200      DEFM      'W - Write buffer to a file'
57 20 2D 20 57 72 69 74 65 20 62 75
66 66 65 72 20 74 6F 20 61 20 66 69
6C 65
55AE 0A      00201      DEFB      10
55AF          00202      DEFM      'X - Exit to DOS'
58 20 2D 20 45 78 69 74 20 74 6F 20
44 4F 53
55BE 0D      00203      DEFB      13
55BF          00204      BAD      DEFM      'Bad command'
42 61 64 20 63 6F 6D 6D 61 6E 64
55CA 0D      00205      DEFB      13
55CB          00206      PAR      DEFM      'Parameter error'
50 61 72 61 6D 65 74 65 72 20 65 72
72 6F 72
55DA 0D      00207      DEFB      13
55DB          00208      ABRKM    DEFM      'Break key value? '
42 72 65 61 6B 20 6B 65 79 20 76 61
6C 75 65 3F 20
55EC 03      00209      DEFB      3
00210 ;
00211 ;      TOGG HIT BIT STRIP
00212 ;
55ED 00      00213      AHIB      NOP
55EE          00214      AHIBM    DEFM      'BIT 7 STRIP IN NOW '
42 49 54 20 37 20 53 54 52 49 50 20
49 4E 20 4E 4F 57 20
5601 03      00215      DEFB      3
5602          00216      BITS     EQU      $
5602 3AED55   00217      LD      A, (AHIB)      ;GET FL
5605 2F        00218      CPL                      ;SWITCH
5606 32ED55   00219      LD      (AHIB), A      ;SAVW
5609 F5        00220      PUSH     AF           ;SAVE
560A 21EE55   00221      LD      HL, AHIBM      ;GET MSG
560D CD045C   00222      CALL     DLINE        ;DIS
5610 213057   00223      LD      HL, AON       ;GET ON
5613 F1        00224      POP      AF          ;GET COND.

```



```

5614 B7      00225      OR      A      ;OFF?
5615 2003    00226      JR      NZ,G36  ;N
5617 213357  00227      LD      HL,AOFF ;GET MSG
561A CD045C  00228 G36   CALL    DLINE ;DIS
561D C3E853  00229      JP      INCMD   ;RET
          00230 ;
          00231 ;      DEFINE BREAK
          00232 ;
5620 CD755B  00233 DEFBRK CALL    POSNC      ;CLR AA
5623 21DB55  00234      LD      HL,ABRKM    ;GET MSG
5626 CD045C  00235      CALL    DLINE      ;dis
5629 211843  00236      LD      HL,4318H
562C 0603    00237      LD      B,3
562E CDBC59  00238      CALL    LIR
5631 380B    00239      JR      C,BKDD      ;BREAK
5633 CD845B  00240      CALL    GETINP     ;GET INP
5636 14      00241      INC      D
5637 15      00242      DEC      D          ;OVER 255?
5638 20E6    00243      JR      NZ,DEFBRK  ;BAD
563A 7B      00244      LD      A,E        ;GET DEF
563B 32AC58  00245      LD      (ABREAK),A ;STOR
563E C3755B  00246 BKDD   JP      POSNC      ;CAA, RET
5641 214F56  00247 OUTMEM LD      HL,OME     ;GET OUT OF MEM
5644 1803    00248      JR      AAL
5646 21CB55  00249 PARERR LD      HL,PAR     ;GET ERR
5649 CD045C  00250 AAL    CALL    DLINE      ;DIS
564C C3E853  00251      JP      INCMD   ;RET
564F      00252 OME     DEFM    'Buffer full'
          42 75 66 66 65 72 20 66 75 6C 6C
565A 0D      00253      DEFB      13
565B      00254 ASTATM DEFM    'Status information'
          53 74 61 74 75 73 20 69 6E 66 6F 72
          6D 61 74 69 6F 6E
566D 0A0D    00255      DEFW      0D0AH
566F 0A      00256 AHIT   DEFB      10
5670 0A      00257      DEFB      10
5671      00258      DEFM      '
          20 20 20 20 20 20 20 20 20 50 72 65
          73 73 20 61 6E 79 20 6B 65 79 20 74
          6F 20 72 65 74 75 72 6E 20 74 6F 20
          6D 65 6E 75
          Press any key to return to menu'
5699 03      00259      DEFB      3
569A      00260 BSTAT   EQU      $
569A CDC901  00261      CALL    1C9H      ;CLS
569D 215B56  00262      LD      HL,ASTATM   ;GET MSG
56A0 CD045C  00263      CALL    DLINE      ;DIS
56A3 11125C  00264      LD      DE,EOP     ;GET BUFF START
56A6 2AB859  00265      LD      HL,(NB)    ;NEXT BYTE
56A9 B7      00266      OR      A          ;CLR C
56AA ED52    00267      SBC      HL,DE     ;GET # BYTE IN BUFFER
56AC 22D557  00268      LD      (INT),HL   ;STOR
56AF 219257  00269      LD      HL,BMSG    ;GET BUF
56B2 0E01    00270      LD      C,1        ;NO LEAD 0
56B4 CDD757  00271      CALL    CI5        ;CONV
56B7 ED5BB859 00272      LD      DE,(NB)    ;NEXT BYTE IN BUFF

```

56BB 2A4940	00273	LD	HL, (4049H)	
56BE B7	00274	OR	A	;GET LAST POS BYTE
56BF ED52	00275	SBC	HL, DE	;NO C
56C1 22D557	00276	LD	(INT), HL	;GET FREE
56C4 0E01	00277	LD	C, 1	;SAVE
56C6 21AB57	00278	LD	HL, BMSG1	;NO LEAD
56C9 CDD757	00279	CALL	CI5	;GET BUFF
56CC 2A8A5A	00280	LD	HL, (ABAUD)	;CONV
56CF 22D557	00281	LD	(INT), HL	
56D2 0E01	00282	LD	C, 1	
56D4 216557	00283	LD	HL, RSTAT	
56D7 CDD57	00284	CALL	CI4	
56DA 3A8C5A	00285	LD	A, (AWORD)	
56DD 327057	00286	LD	(ARQ1), A	
56E0 3A8D5A	00287	LD	A, (ASTOP)	
56E3 327C57	00288	LD	(ARQ2), A	
56E6 216557	00289	LD	HL, RSTAT	
56E9 CD045C	00290	CALL	DLINE	;DISPLAY
56EC 3A8E5A	00291	LD	A, (APAR)	
56EF 21C357	00292	LD	HL, APEVEN	
56F2 FE45	00293	CP	'E'	
56F4 280A	00294	JR	Z, JJJ	
56F6 21C857	00295	LD	HL, APODD	
56F9 FE4F	00296	CP	'O'	
56FB 2803	00297	JR	Z, JJJ	
56FD 21CC57	00298	LD	HL, APNONE	
5700 CD045C	00299 JJJ	CALL	DLINE	;DIS
5703 218A57	00300	LD	HL, ARQ3	
5706 CD045C	00301	CALL	DLINE	;DISPLAY TXT
5709 216F56	00302	LD	HL, AHIT	;get return msg
570C CD045C	00303	CALL	DLINE	;dis
570F CD4900	00304	CALL	49H	;WAIT FOR KEY
5712 C3DC53	00305	JP	MAIN	;RET
5715	00306 ;			
5715 3AB759	00307 ECHO	EQU	\$	
5718 EEFF	00308	LD	A, (HDX)	;GET ECHO SWITCH
571A 32B759	00309	XOR	255	;SWTICH
571D F5	00310	LD	(HDX), A	;SAVE
571E 213757	00311	PUSH	AF	;SAVE
5721 CD045C	00312	LD	HL, AECHO	
5724 213357	00313	CALL	DLINE	;DIS
5727 F1	00314	LD	HL, AOFF	;GET OFF MSG
5728 2803	00315	POP	AF	;GET STAT
572A 213057	00316	JR	Z, DIST	;GO IF OFF
572D C3045C	00317	LD	HL, AON	
5730 6F6E	00318 DIST	JP	DLINE	;DIS & RET
5732 0D	00319 AON	DEFM	'on'	
5733 6F6666	00320	DEFB	13	
5736 0D	00321 AOFF	DEFM	'off'	
5737	00322	DEFB	13	
	00323 AECHO	DEFM	'Echo is now '	
	45 63 68 6F 20 69 73 20 6E 6F 77 20			
5743 03	00324	DEFB	3	
5744 21125C	00325 CLEAR	LD	HL, EOP	;START OF BUFF
5747 22B859	00326	LD	(NB), HL	;NEXT BYT

```

574A 22BA59 00327 LD (CB),HL ;CUR BYT
574D 215657 00328 LD HL,BCLE ;GET MSG
5750 CD045C 00329 CALL DLINE ;DIS
5753 C3E853 00330 JP INCMD ;CONT
00331 ;
5756 00332 BCLE DEFM 'Buffer cleared'
42 75 66 66 65 72 20 63 6C 65 61 72
65 64
5764 0D 00333 DEFB 13
5765 00334 RSTAT DEFM '0000 Baud, '
30 30 30 30 20 42 61 75 64 2C 20
5770 00335 ARQ1 DEFM 'X Bit word, '
58 20 42 69 74 20 77 6F 72 64 2C 20
577C 00336 ARQ2 DEFM 'X stop bits, '
58 20 73 74 6F 70 20 62 69 74 73 2C
20
5789 03 00337 DEFB 3
578A 00338 ARQ3 DEFM ' Parity'
20 50 61 72 69 74 79
5791 0A 00339 DEFB 10
5792 00340 BMSG DEFM '00000 Characters stored,
30 30 30 30 30 20 43 68 61 72 61 63
74 65 72 73 20 73 74 6F 72 65 64 2C
20
57AB 00341 BMSG1 DEFM '00000 Free buffer bytes'
30 30 30 30 30 20 46 72 65 65 20 62
75 66 66 65 72 20 62 79 74 65 73
57C2 03 00342 DEFB 3
57C3 4576656E 00343 APEVEN DEFM 'Even'
57C7 03 00344 DEFB 3
57C8 4F6464 00345 APODD DEFM 'Odd'
57CB 03 00346 DEFB 3
57CC 00347 APNONE DEFM 'Disabled'
44 69 73 61 62 6C 65 64
57D4 03 00348 DEFB 3
00349 ;
00350 ; PUTS DECIMAL EQU OF (INT) VALUE.
00351 ; ENTRY : HL->DEST, C=0 LEADING ZEROS
00352 ;
57D5 0000 00353 INT DEFW 0
57D7 111027 00354 CI5 LD DE,10000
57DA CDF457 00355 CALL CVI
57DD 11E803 00356 CI4 LD DE,1000
57E0 CDF457 00357 CALL CVI
57E3 116400 00358 CI3 LD DE,100
57E6 CDF457 00359 CALL CVI
57E9 110A00 00360 CI2 LD DE,10
57EC CDF457 00361 CALL CVI
57EF 3AD557 00362 LD A,(INT)
57F2 1827 00363 JR LOP4
57F4 C5 00364 CVI PUSH BC
57F5 0600 00365 LD B,0
57F7 E5 00366 PUSH HL
57F8 2AD557 00367 LD HL,(INT)
57FB B7 00368 LOP OR A

```

Term Program

57FC ED52	00369	SBC	HL,DE	
57FE 3807	00370	JR	C,LOP3	
5800 F5	00371	PUSH	AF	
5801 04	00372	INC	B	
5802 F1	00373	POP	AF	
5803 2803	00374	JR	Z,LOP2	
5805 18F4	00375	JR	LOP	
5807 19	00376 LOP3	ADD	HL,DE	
5808 22D557	00377 LOP2	LD	(INT),HL	
580B 78	00378	LD	A,B	
580C E1	00379	POP	HL	
580D C1	00380	POP	BC	
580E B7	00381	OR	A	
580F 2804	00382	JR	Z,LOP4A	
5811 0E00	00383	LD	C,0	
5813 1806	00384	JR	LOP4	
5815 79	00385 LOP4A	LD	A,C	
5816 B7	00386	OR	A	
5817 2802	00387	JR	Z,LOP4	
5819 3ED0	00388	LD	A,-30H&255	
581B C630	00389 LOP4	ADD	A,30H	
581D 77	00390	LD	(HL),A	
581E 23	00391	INC	HL	
581F C9	00392	RET		
5820 1C	00393 ATERM	DEFB	28	
5821 1F	00394	DEFB	31	
5822 0E	00395	DEFB	14	
5823	00396	DEFM	'Smart terminal mode - Press <SHIFT>-<BREAK> to escape'	
			53 6D 61 72 74 20 74 65 72 6D 69 6E	
			61 6C 20 6D 6F 64 65 20 2D 20 50 72	
			65 73 73 20 3C 53 48 49 46 54 3E 2D	
			3C 42 52 45 41 4B 3E 20 74 6F 20 65	
			73 63 61 70 65	
5858 0A	00397	DEFB	10	
5859 0D	00398	DEFB	13	
	00399 ;			
	00400 ;	TERM ROUTINE		
	00401 ;			
585A	00402 TERM	EQU	\$	
585A 212058	00403	LD	HL,ATERM	;GET MSG
585D CD045C	00404	CALL	DLINE	;DIS
5860 1838	00405	JR	NDI	;CON
5862	00406 TMAIN	EQU	\$	
5862 CD2353	00407	CALL	INPUT	;FECTH UART INP
5865 3803	00408	JR	C,NIP	;NO INPUT
5867 CDAE58	00409	CALL	OUTDEV	;OUTPUT TO DO, BUFFER
586A CD2B00	00410 NIP	CALL	2BH	;GET A KEY
586D B7	00411	OR	A	;SET F
586E 28F2	00412	JR	Z,TMAIN	;NO KEYINP
5870 FE01	00413	CP	1	;BREAK
5872 282C	00414	JR	Z,CSW	;CHECK
5874 FE1F	00415	CP	31	;CLEAR <CTRL>
5876 28EA	00416	JR	Z,TMAIN	;Y
5878 4F	00417	LD	C,A	;XFR CHAR
5879 3A4038	00418	LD	A,(3840H)	;GET CLR

587C CB4F	00419	BIT	1,A	;CTRL?
587E 79	00420	LD	A,C	;GET CHAR
587F 280A	00421	JR	Z,KLW	;NO
5881 E61F	00422	AND	31	;MAKE CTRL CHR
5883 FEL1	00423	CP	'Q'-64	;SPOOL INPUT?
5885 2814	00424	JR	Z,STI	;YES
5887 FEL0	00425	CP	'P'-64	;SPOOL OFF?
5889 280F	00426	JR	Z,NDI	;YES
588B CD5453	00427	CALL	OUTPUT	;SEND
588E 3AB759	00428	LD	A,(HDX)	;HALF DUPE?
5891 B7	00429	OR	A	
5892 28CE	00430	JR	Z,TMAIN	;NO
5894 79	00431	LD	A,C	;YES
5895 CD3300	00432	CALL	33H	;DIS
5898 18C8	00433	JR	TMAIN	;LOOP
589A AF	00434	NDI	XOR A	;SPOOL OFF
589B 32AD58	00435	STI	LD (SPF),A	;STOR SPOOL FLG
589E 18C2	00436	JR	TMAIN	;CONT
58A0 3A8038	00437	CSW	LD A,(3880H)	;GET CTRL BYTE
58A3 1F	00438	RRA		;SHIFT IN C
58A4 DADC53	00439	JP	C,MAIN	;ABORT TERM MODE
58A7 3AAC58	00440	LD	A,(ABREAK)	;GET BREAK VAL
58AA 18DF	00441	JR	KLW	;CONT
58AC 01	00442	ABREAK	DEFB 1	;CHR 1
58AD 00	00443	SPF	NOP	;MEM SPOOL FLAG
58AE	00444	OUTDEV	EQU \$	
58AE 4F	00445	LD	C,A	;SAVE CHAR
58AF CD3300	00446	CALL	33H	;DIS
58B2 3AAD58	00447	LD	A,(SPF)	;GET SPOOL
58B5 B7	00448	OR	A	;CHECK
58B6 79	00449	LD	A,C	;GET CHAR
58B7 C8	00450	RET	Z	;NO SPOOL
58B8 ED5BB859	00451	LD	DE,(NB)	;GET CHAR
58BC 2A4940	00452	LD	HL,(4049H)	;GET HI ME
58BF ED52	00453	SBC	HL,DE	;OK?
58C1 3808	00454	JR	C,NIH	;TO HI
58C3 79	00455	LD	A,C	;GET CHAR
58C4 12	00456	LD	(DE),A	;STORE CHAR
58C5 13	00457	INC	DE	;BMP
58C6 ED53B859	00458	LD	(NB),DE	;STOR
58CA C9	00459	RET		;CONT
58CB 3E8F	00460	NIH	LD A,143	;BLOCK
58CD CD3300	00461	CALL	33H	;DIS
58D0 C36258	00462	JP	TMAIN	;CONT
	00463	;		
	00464	;	PRINT/DISPLAY/SEND BUFFER	
	00465	;		
58D3 115453	00466	SEND	LD DE,OUTPUT	;OUTPUT ROUTINE
58D6 180B	00467	JR	SAL	;CONT
58D8 3E33	00468	DISP	LD A,33H	;DISPLAY ADR
58DA 01	00469	DEFB	1	
58DB 3E3B	00470	PRINT	LD A,3BH	;PRINTER ADR
58DD 01	00471	DEFB	1	
58DE 3E1B	00472	SAVE	LD A,1BH	;WRITE FILE
58E0 5F	00473	LD	E,A	;GET LSB

Term Program

```

58E1 1600      00474      LD      D,0      ;0
58E3 ED532A59 00475 SAL      LD      (OQ+1),DE ;STOR
58E7 4F        00476      LD      C,A      ;SAVE DEVICE
58E8 AF        00477      XOR      A        ;CLR
58E9 E5        00478      PUSH     HL      ;SAVE LINE
58EA 21125C    00479      LD      HL,EOP    ;GET BUFF
58ED ED5BB859 00480      LD      DE,(NB)    ;GET NEXT
58F1 DF        00481      RST      18H      ;COMPARE
58F2 283A      00482      JR      Z,PRER    ;NO BUFF CONT
58F4 E3        00483      EX      (SP),HL   ;HL=FS. PUSH EOP
58F5 CD5159    00484      CALL    SAVEI    ;OPEN FILE IF SAVE
58F8 E1        00485      POP      HL      ;GET EOP
58F9 CD4459    00486 LP      CALL    MEMS     ;TEST IF END
58FC 2015      00487      JR      NZ,JEI    ;NOT DONE
58FE 3A2A59    00488 ENDD    LD      A,(OQ+1) ;GET DEVICE
5901 FE1B      00489      CP      1BH      ;FILE OUTPUT?
5903 200B      00490      JR      NZ,HR2    ;NO
5905 110053    00491      LD      DE,FCB    ;GET FCB
5908 CD2844    00492      CALL    4428H    ;CLOSE FILE
590B 2803      00493      JR      Z,HR2    ;NO ERR
590D CD7459    00494      CALL    ERROR    ;DIS ERR
5910 C3E853    00495 HR2    JP      INCMD    ;RET
5913 CD2B00    00496 JEI      CALL    2BH      ;GET KEY
5916 FE01      00497 JEI1    CP      1        ;BREAK?
5918 CAF58     00498      JP      Z,ENDD    ;ABORT
591B FE60      00499      CP      96      ;SHIFT @?
591D 2005      00500      JR      NZ,JEI2    ;NO PAUSE
591F CD4900    00501      CALL    49H      ;WAIT ON KEY
5922 18F2      00502      JR      JEI1      ;GO CHECK KEY
5924 7E        00503 JEI2    LD      A,(HL)    ;GET BYTE
5925 23        00504      INC      HL      ;BMP BUFF
5926 110053    00505      LD      DE,FCB    ;GET FILE FCB
5929 CD3300    00506 OQ      CALL    33H      ;DIS/PR/WRITE
592C 18CB      00507      JR      LP        ;LOOP
                    00508 ;
592E 213759    00509 PRER    LD      HL,PMH     ;GET MSG
5931 CD045C    00510      CALL    DLINE    ;DIS
5934 C3E853    00511      JP      INCMD    ;RET
5937           00512 PMH    DEFM    'Buffer empty'
                    42 75 66 66 65 72 20 65 6D 70 74 79
5943 0D        00513      DEFB      13
                    00514 ;
5944           00515 MEMS    EQU      $        ;Z=EO BUFFER
5944 ED5BB859 00516      LD      DE,(NB)    ;GET CUR
5948 DF        00517      RST      18H      ;COMP
5949 C9        00518      RET
594A           00519 HIGH    EQU      $        ;Z=PAST HI, ONE BYTE
594A ED5B4940 00520      LD      DE,(4049H) ;GET HI
594E 13        00521      INC      DE      ;BMP
594F DF        00522      RST      18H      ;COMP
5950 C9        00523      RET
                    00524 ;
                    00525 ; OPEN FCB FOR SAVE
                    00526 ;
5951           00527 SAVEI    EQU      $

```

```

5951 79      00528      LD      A,C      ;GET DEVICE
5952 FELB    00529      CP      1BH      ;OUTPUT TO FILE?
5954 C0      00530      RET      NZ      ;NO
5955 23      00531      INC      HL      ;BMP PTR
5956 CD6953  00532      CALL     NEXT     ;GET NEXT
5959 110053  00533      LD      DE,FCB    ;GET FCB
595C CD1C44  00534      CALL     441CH    ;MOVE FS
595F 210052  00535      LD      HL,BUFFER ;GET I/O BUFFER
5962 0600    00536      LD      B,0      ;LRL=256
5964 CD2044  00537      CALL     4420H    ;OPEN/INIT
5967 C8      00538      RET      Z      ;NO ERR
5968 CD7459  00539      CALL     ERROR    ;DISPLAY ERR
596B C3E853  00540      JP      INCMD    ;RET
596E CD7459  00541 ERRX   CALL     ERROR    ;
5971 C3E853  00542      JP      INCMD    ;
5974 F6C0    00543 ERROR  OR      0C0H     ;RET RET
5976 CD0944  00544      CALL     4409H    ;DIS ERROR
5979 C9      00545      RET
          00546 ;
          00547 ;      LOAD ROUTINE
          00548 ;
597A      00549 LOAD    EQU      $
597A 23      00550      INC      HL      ;BMP
597B CD6953  00551      CALL     NEXT     ;POSN
597E CA4656  00552      JP      Z,PARERR  ;NO F/S
5981 110053  00553      LD      DE,FCB    ;GET FCB
5984 CD1C44  00554      CALL     441CH    ;MOVE FS
5987 0600    00555      LD      B,0      ;LRL=256
5989 210052  00556      LD      HL,BUFFER ;BUFF
598C CD2444  00557      CALL     4424H    ;OPEN FILE
598F C26E59  00558      JP      NZ,ERRX   ;ERR
5992 2AB859  00559      LD      HL,(NB)   ;NEXT BYTE LOC
5995 CD4A59  00560 LR2   CALL     HIGH     ;TEST FOR OK
5998 CA4156  00561      JP      Z,OUTMEM  ;OUT OF MEM
599B 110053  00562      LD      DE,FCB    ;GET FCB
599E CD1300  00563      CALL     13H      ;GET BYTE
59A1 C2AB59  00564      JP      NZ,EML    ;ERR
59A4 77      00565      LD      (HL),A    ;STORE BYTE
59A5 23      00566      INC      HL      ;STOR
59A6 22B859  00567      LD      (NB),HL   ;UPD NEXT BYTE
59A9 18EA    00568      JR      LR2      ;LOOP
59AB FE1C    00569 EML   CP      28      ;EOF?
59AD 2805    00570      JR      Z,EMN    ;OK
59AF FE1D    00571      CP      29      ;PAST?
59B1 C26E59  00572      JP      NZ,ERRX   ;ERR
59B4 C3E853  00573 EMN   JP      INCMD    ;RET
59B7 00      00574 HDX   NOP
59B8 125C    00575 NB   DEFW     EOP     ;HALF DUPLEX FLAG
59BA 125C    00576 CB   DEFW     EOP     ;NEXT BYT IN BUFF
          00577 ;      ;CURR BYT WHILE DIS/PR
          00578 ;      LINE INPUT ROUTINE
          00579 ;
59BC      00580 LIR   EQU      $
59BC 3E0E    00581      LD      A,14
59BE CD3300  00582      CALL     33H

```

Term Program

59C1	E5	00583		PUSH	HL	;SAVE BUFF
59C2	48	00584		LD	C,B	;XFER LIMIT
59C3	0600	00585		LD	B,0	;RESET COUNTER
59C5	CD4900	00586	LIRL	CALL	49H	;GET CHR
59C8	FE01	00587		CP	1	;BRK?
59CA	283B	00588		JR	Z,BRKL'	;Y
59CC	FE0D	00589		CP	13	;DONE?
59CE	2838	00590		JR	Z,LDON	;Y
59D0	FE08	00591		CP	8	;RUB?
59D2	2826	00592		JR	Z,RUB	;Y
59D4	FELF	00593		CP	31	;DEL LINE?
59D6	2816	00594		JR	Z,LIRD	;Y
59D8	5F	00595		LD	E,A	;XFER CHR
59D9	78	00596		LD	A,B	;GET COUNT
59DA	B9	00597		CP	C	;MAX?
59DB	28E8	00598		JR	Z,LIRL	;Y
59DD	7B	00599		LD	A,E	;GET CHR
59DE	FE20	00600		CP	32	;OK?
59E0	38E3	00601		JR	C,LIRL	;N
59E2	FE7B	00602		CP	'z'+1	;TOO HI?
59E4	30DF	00603		JR	NC,LIRL	;Y
59E6	77	00604		LD	(HL),A	;DIS
59E7	CD3300	00605		CALL	33H	;DIS
59EA	23	00606		INC	HL	;BMP
59EB	04	00607		INC	B	;REGS
59EC	18D7	00608		JR	LIRL	;CONT
59EE	78	00609	LIRD	LD	A,B	;GET COUNT
59EF	B7	00610		OR	A	;BEG OF LIN?
59F0	28D3	00611		JR	Z,LIRL	;Y
59F2	3E08	00612		LD	A,8	;BKSP
59F4	CD3300	00613		CALL	33H	;DIS
59F7	05	00614		DEC	B	;DEC COUNT
59F8	18F4	00615		JR	LIRD	;CONT
59FA	78	00616	RUB	LD	A,B	;GET CNT
59FB	B7	00617		OR	A	;AT BEG?
59FC	28C7	00618		JR	Z,LIRL	;Y
59FE	3E08	00619		LD	A,8	;BKSP
5A00	CD3300	00620		CALL	33H	;DIS
5A03	2B	00621		DEC	HL	;BACKUP
5A04	05	00622		DEC	B	;DEC COUNT
5A05	18BE	00623		JR	LIRL	;CONT
5A07	37	00624	BRKL	SCF		;FOR BREAK
5A08	360D	00625	LDON	LD	(HL),13	;C/R
5A0A	E1	00626		POP	HL	;GET BUFF
5A0B	F5	00627		PUSH	AF	;SAVE CF
5A0C	3E0F	00628		LD	A,15	;CUR OFF
5A0E	CD3300	00629		CALL	33H	
5A11	3E0D	00630		LD	A,13	;C/R
5A13	CD3300	00631		CALL	33H	;DIS
5A16	F1	00632		POP	AF	;GET CF
5A17	C9	00633		RET		
5A18	0000	00634	CURL0	DEFW	0	;CUR LOC
5A1A		00635	ARSL	DEFM	'Baud (110,150,300,600,1200,2400,4800,9600) ? '	

42 61 75 64 20 28 31 31 30 2C 31 35
30 2C 33 30 30 2C 36 30 30 2C 31 32


```

30 30 2C 32 34 30 30 2C 34 38 30 30
2C 39 36 30 30 29 20 20 3F 20
5A48 03      00636      DEFB      3
5A49          00637 ARS2      DEFM      'Word length (5,6,7,8) ? '
57 6F 72 64 20 6C 65 6E 67 74 68 20
28 35 2C 36 2C 37 2C 38 29 20 20 3F
20
5A62 03      00638      DEFB      3
5A63          00639 ARS3      DEFM      'Stop bits (1,2) ? '
53 74 6F 70 20 62 69 74 73 20 28 31
2C 32 29 20 20 3F 20
5A76 03      00640      DEFB      3
5A77          00641 ARS4      DEFM      'Parity (E,O,N) ? '
50 61 72 69 74 79 20 28 45 2C 4F 2C
4E 29 20 20 3F 20
5A89 03      00642      DEFB      3
5A8A 0000     00643 ABAUD     DEFW      0 ;CURRENT
5A8C 00      00644 AWORD     NOP
5A8D 00      00645 ASTOP     NOP
5A8E 00      00646 APAR      NOP
5A8F 6E00     00647 ABRATE     DEFW      110
5A91 22      00648          DEFM      ' '
5A92 9600     00649          DEFW      150
5A94 44      00650          DEFB      44H
5A95 2C01     00651          DEFW      300
5A97 55      00652          DEFB      55H
5A98 5802     00653          DEFW      600
5A9A 66      00654          DEFB      66H
5A9B B004     00655          DEFW      1200
5A9D 77      00656          DEFB      77H
5A9E 6009     00657          DEFW      2400
5AA0 AA      00658          DEFB      0AAH
5AA1 C012     00659          DEFW      4800
5AA3 CC      00660          DEFB      0CCH
5AA4 8025     00661          DEFW      9600
5AA6 EE      00662          DEFB      0EEH
5AA7 0000     00663          DEFW      0 ;TERM
5AA9 00      00664 AWORDL    DEFB      0
5AAA 4020     00665          DEFM      '@ '
5AAC 60      00666          DEFB      60H
00667 ;
00668 ; SET RS-232 PARAMETERS
00669 ;
5AAD          00670 RESET     EQU      $
5AAD CD755B   00671          CALL     POSNC ;POSN CUR
5AB0 211A5A   00672          LD      HL,ARS1 ;BAUD MSG
5AB3 CD045C   00673          CALL     DLINE ;DIS
5AB6 211843   00674          LD      HL,4318H ;IN BUF
5AB9 0604     00675          LD      B,4 ;MAX LEN
5ABB CDBC59   00676          CALL     LIR ;GET INP
5ABE 38ED     00677          JR      C,RESET ;BREAK HIT
5AC0 CD845B   00678          CALL     GETINP ;GET INP
5AC3 218F5A   00679          LD      HL,ABRATE ;GET BAUD TABLE
5AC6 4E      00680 BST      LD      C,(HL) ;GET BAUD# LSB
5AC7 23      00681          INC     HL

```

Term Program

5AC8 46	00682	LD	B, (HL)	;MSB
5AC9 23	00683	INC	HL	
5ACA 23	00684	INC	HL	
5ACB 78	00685	LD	A,B	;IS BC 0?
5ACC B1	00686	OR	C	
5ACD 28DE	00687	JR	Z,RESET	;Y, BAD INP
5ACF 7A	00688	LD	A,D	
5AD0 B8	00689	CP	B	
5AD1 20F3	00690	JR	NZ,BST	;TRY NEXT
5AD3 7B	00691	LD	A,E	
5AD4 B9	00692	CP	C	
5AD5 20EF	00693	JR	NZ,BST	;TRY NEXT
5AD7 2B	00694	DEC	HL	;POSN TO BUAD
5AD8 7E	00695	LD	A, (HL)	
5AD9 322153	00696	LD	(ABAUDV),A	;STOR VAL
5ADC ED538A5A	00697	LD	(ABAUD),DE	;STOR #
	00698 ;			
	00699 ;			
	00700 ;			
		GET WORD LEN		
5AE0	00701	EQU	\$	
5AE0 CD755B	00702	CALL	POSNC	;POSN CUR
5AE3 21495A	00703	LD	HL,ARS2	;GET MSG
5AE6 CD045C	00704	CALL	DLINE	;DIS
5AE9 211843	00705	LD	HL,4318H	;IN BUF
5AEC 0601	00706	LD	B,1	;1 CHR
5AEE CDBC59	00707	CALL	LIR	;GET INP
5AF1 38ED	00708	JR	C,GWL	;BREAK
5AF3 7E	00709	LD	A, (HL)	;GET CHR
5AF4 328C5A	00710	LD	(AWORD),A	;STOR
5AF7 D635	00711	SUB	35H	;GET REL LEN
5AF9 38E5	00712	JR	C,GWL	;TOO LO
5AFB FE04	00713	CP	4	;TO HI?
5AFD 30E1	00714	JR	NC,GWL	;Y
5AFF 21A95A	00715	LD	HL,AWORDL	;GET TABLE
5B02 1600	00716	LD	D,0	;0
5B04 5F	00717	LD	E,A	;FOR AD
5B05 19	00718	ADD	HL,DE	;PT TO BYTE
5B06 7E	00719	LD	A, (HL)	;GET LEN BITS
5B07 322053	00720	LD	(ACURS),A	;STOR STAT
	00721 ;			
	00722 ;			
	00723 ;			
		GET STOP BITS		
5B0A	00724	EQU	\$	
5B0A CD755B	00725	CALL	POSNC	;POSN CUR
5B0D 21635A	00726	LD	HL,ARS3	;STOP MSG
5B10 CD045C	00727	CALL	DLINE	;DIS
5B13 211843	00728	LD	HL,4318H	;INP BUF
5B16 0601	00729	LD	B,1	;1 CHR
5B18 CDBC59	00730	CALL	LIR	;INP
5B1B 38ED	00731	JR	C,GSB	;REDO
5B1D 7E	00732	LD	A, (HL)	;GET CHR
5B1E 328D5A	00733	LD	(ASTOP),A	;STOR
5B21 D631	00734	SUB	31H	;MAKE REL
5B23 38E5	00735	JR	C,GSB	;BAD INP
5B25 FE02	00736	CP	2	;TO HI?

```

5B27 30E1      00737      JR      NC,GSB          ;Y
5B29 0E00      00738      LD      C,0            ;NO STOP
5B2B B7        00739      OR      A              ;1 STOP?
5B2C 2802      00740      JR      Z,YLSB         ;Y
5B2E 0E10      00741      LD      C,10H          ;2 STOP
5B30 3A2053    00742 YLSB LD      A,(ACURS),      ;GET BYTE
5B33 B1        00743      OR      C              ;ADD STOP BITS
5B34 322053    00744      LD      (ACURS),A       ;STOR
                    00745 ;
                    00746 ;      GET PARITY
                    00747 ;
5B37           00748 GPB   EQU      $
5B37 CD755B    00749      CALL   POSNC
5B3A 21775A    00750      LD      HL,ARS4
5B3D CD045C    00751      CALL   DLINE          ;DISP
5B40 211843    00752      LD      HL,4318H
5B43 0601      00753      LD      B,1
5B45 CDBC59    00754      CALL   LIR
5B48 38ED      00755      JR      C,GPB
5B4A 7E        00756      LD      A,(HL)
5B4B CBAF      00757      RES     5,A           ;MAKE U/C
5B4D 328E5A    00758      LD      (APAR),A       ;STOR
5B50 0E80      00759      LD      C,80H
5B52 FE45      00760      CP      'E'           ;EVEN?
5B54 280C      00761      JR      Z,YPX         ;Y
5B56 0E00      00762      LD      C,0
5B58 FE4F      00763      CP      'O'
5B5A 2806      00764      JR      Z,YPX         ;ODD
5B5C 0E08      00765      LD      C,8
5B5E FE4E      00766      CP      'N'           ;NO PAR?
5B60 20D5      00767      JR      NZ,GPB        ;INP ERR
5B62 3A2053    00768 YPX LD      A,(ACURS)      ;GET STAT
5B65 B1        00769      OR      C              ;ADD PARITY
5B66 F607      00770      OR      7           ;TURN ON CTRL
5B68 322053    00771      LD      (ACURS),A       ;STOR
5B6B 4F        00772      LD      C,A           ;GET CTRL WORD
5B6C 3A2153    00773      LD      A,(ABAUDV)    ;GET BAUD VAL
5B6F CD6153    00774      CALL   INITRS        ;INIT RS-232
5B72 C3755B    00775      JP      POSNC         ;CAA
                    00776 ;
5B75           00777 POSNC EQU      $
5B75 2A185A    00778      LD      HL,(CURLO)      ;GET INP POSN
5B78 114000    00779      LD      DE,64        ;# CHRS LIN
5B7B 19        00780      ADD     HL,DE         ;BMP
5B7C 222040    00781      LD      (4020H),HL    ;POSN CUR
5B7F 3E1F      00782      LD      A,31        ;CLR ALL AFTER
5B81 C33300    00783      JP      33H          ;DO IT
                    00784 ;
                    00785 ;
                    00786 ;      GETINP - DECIMAL & HEX INPUT DECODER
                    00787 ;      ENTRY HL-> INP, EXIT DE=NUM
                    00788 ;      CF = ENTRY WAS TOO HI
                    00789 ;
5B84 E5        00790 GETINP PUSH   HL           ;SAVE LINE
5B85 110000    00791      LD      DE,0         ;RESET

```

Term Program

5B88	7E	00792	GIN	LD	A, (HL)	;GET CHR
5B89	FE0D	00793		CP	13	;DUN?
5B8B	2003	00794		JR	NZ,GIN2	;N
5B8D	E3	00795	GIN3	EX	(SP),HL	;LINE ON STK
5B8E	E1	00796		POP	HL	;GET LIN
5B8F	C9	00797		RET		;DUN
5B90	23	00798	GIN2	INC	HL	;BMP LIN
5B91	FE20	00799		CP	32	;SPACE?
5B93	28F8	00800		JR	Z,GIN3	;DUN
5B95	FE30	00801		CP	'0'	;NUM?
5B97	3819	00802		JR	C,HEXINP	;N
5B99	FE3A	00803		CP	':'	;NUM?
5B9B	3015	00804		JR	NC,HEXINP	;N
5B9D	D630	00805		SUB	30H	;CONV BIN
5B9F	E5	00806		PUSH	HL	;SAVE LIN
5BA0	62	00807		LD	H,D	;HL=ACCUM
5BA1	6B	00808		LD	L,E	
5BA2	29	00809		ADD	HL,HL	;HL=HL*10
5BA3	29	00810		ADD	HL,HL	
5BA4	19	00811		ADD	HL,DE	
5BA5	29	00812		ADD	HL,HL	
5BA6	3805	00813		JR	C,GINX	;O/V ERR
5BA8	1600	00814		LD	D,0	;0
5BAA	5F	00815		LD	E,A	;GET CUR DIGIT
5BAB	19	00816		ADD	HL,DE	;ADD DIGIT
5BAC	EB	00817		EX	DE,HL	;DE=ACCUM
5BAD	E1	00818	GINX	POP	HL	;RESTOR LIN
5BAE	30D8	00819		JR	NC,GIN	;OK
5BB0	18DB	00820		JR	GIN3	;OVERFLOW
		00821				
5BB2	E1	00822	HEXINP	POP	HL	;GET INP START
5BB3	110000	00823		LD	DE,0	;RESET ACCUM
5BB6	7E	00824	HEX	LD	A, (HL)	;GET DIGIT
5BB7	FE0D	00825		CP	13	;DUN?
5BB9	C8	00826		RET	Z	;Y
5BBA	FE20	00827		CP	32	;DUN?
5BBC	C8	00828		RET	Z	;Y
5BBD	23	00829		INC	HL	;BMP
5BBE	FE48	00830		CP	'H'	; 'HEX' SIG?
5BC0	28F4	00831		JR	Z,HEX	;IGNOR
5BC2	FE30	00832		CP	'0'	;OK?
5BC4	D8	00833		RET	C	;N
5BC5	FE47	00834		CP	'G'	;OK?
5BC7	3F	00835		CCF		;INV CF
5BC8	D8	00836		RET	C	;N
5BC9	FE3A	00837		CP	':'	;NUM?
5BCB	3805	00838		JR	C,HEX1	;Y
5BCD	FE41	00839		CP	'A'	;HEX?
5BCF	D8	00840		RET	C	;BAD
5BD0	D607	00841		SUB	7	;SUB AMT
5BD2	D630	00842	HEX1	SUB	30H	;MAKE BIN
5BD4	EB	00843		EX	DE,HL	;HL=ACCUM
5BD5	F5	00844		PUSH	AF	;SAVE BIN
5BD6	7C	00845		LD	A,H	;GET MSB
5BD7	E6F0	00846		AND	0F0H	;TEST IF MAX AL

5BD9 37	00847	SCF		;SET C
5BDA 2803	00848	JR	Z,HEX7	;OK
5BDC EB	00849	EX	DE,HL	;RESTOR HL, LIN
5BDD F1	00850	POP	AF	;KILL
5BDE C9	00851	RET		
5BDF 29	00852 HEX7	ADD	HL,HL	;SHIFT OVER 4X
5BE0 29	00853	ADD	HL,HL	;HL=HL*16
5BE1 29	00854	ADD	HL,HL	
5BE2 29	00855	ADD	HL,HL	
5BE3 F1	00856	POP	AF	;GET DIGIT
5BE4 B5	00857	OR	L	;ADD NEW DIGIT
5BE5 6F	00858	LD	L,A	;HL=ACCUM
5BE6 EB	00859	EX	DE,HL	;DE=ACCUM
5BE7 18CD	00860	JR	HEX	;CONT
	00861 ;			
	00862 ;	SEND ASCII SET		
	00863 ;			
5BE9	00864 ASCII	EQU	\$	
5BE9 3E0D	00865	LD	A,13	
5BEB CD5453	00866	CALL	OUTPUT	
5BEE 3E20	00867	LD	A,32	;SPACE
5BF0 CD5453	00868 ASC	CALL	OUTPUT	;SEND OUT
5BF3 0C	00869	INC	C	;BMP VAL
5BF4 CD2B00	00870	CALL	2BH	;GET KEY
5BF7 FE01	00871	CP	1	;BREAK?
5BF9 C8	00872	RET	Z	;Y
5BFA 79	00873	LD	A,C	;GET CHR
5BFB FE5B	00874	CP	'Z'+1	;DONE?
5BFD 28EA	00875	JR	Z,ASCII	;Y
5BFF 18EF	00876	JR	ASC	;CONT
	00877 ;			
5C01 C30052	00878	JP	5200H	
	00879 ;			
5C04	00880 DLINE	EQU	\$;OUTPUT LINE TO VIDEO
5C04 7E	00881	LD	A,(HL)	;GET CHR
5C05 FE03	00882	CP	3	;END OF TEXT?
5C07 C8	00883	RET	Z	;YES
5C08 CD3300	00884	CALL	33H	;DISP
5C0B 7E	00885	LD	A,(HL)	;GET CHR
5C0C FE0D	00886	CP	13	;END OF TEXT?
5C0E C8	00887	RET	Z	;YES
5C0F 23	00888	INC	HL	;BMP TXT PTR
5C10 18F2	00889	JR	DLINE	;CONT DISP
5C12	00890 EOP	EQU	\$	
5C12	00891	END	START	
00000 total errors				

Appendix II

FD1771-01 Data Sheets	197
FD1771-01 Application Notes	217
FD179X-02 Data Sheets	225
FD179X Application Notes	247

Min har en
1771 og en 1791
SD DD

WESTERN DIGITAL

C O R P O R A T I O N

FD1771-01 Floppy Disk Formatter/Controller

FD1771-01

FEATURES

- SOFT SECTOR FORMAT COMPATIBILITY
- AUTOMATIC TRACK SEEK WITH VERIFICATION
- READ MODE
 - Single/Multiple Sector Write with Automatic Sector Search or Entire Track Read
 - Selectable 128 Byte or Variable Length Sector
- WRITE MODE
 - Single/Multiple Sector Write with Automatic Sector Search
 - Entire Track Write for Diskette Formatting
- PROGRAMMABLE CONTROLS
 - Selectable Track-to-Track Stepping Time
 - Selectable Head Settling and Head Engage Times
 - Selectable Three Phase or Step and Direction and Head Positioning Motor Controls
- SYSTEM COMPATIBILITY
 - Double Buffering of Data 8-Bit Bi-Directional Bus for Data, Control and Status
 - DMA or Programmed Data Transfers
 - All Inputs and Outputs are TTL Compatible

APPLICATIONS

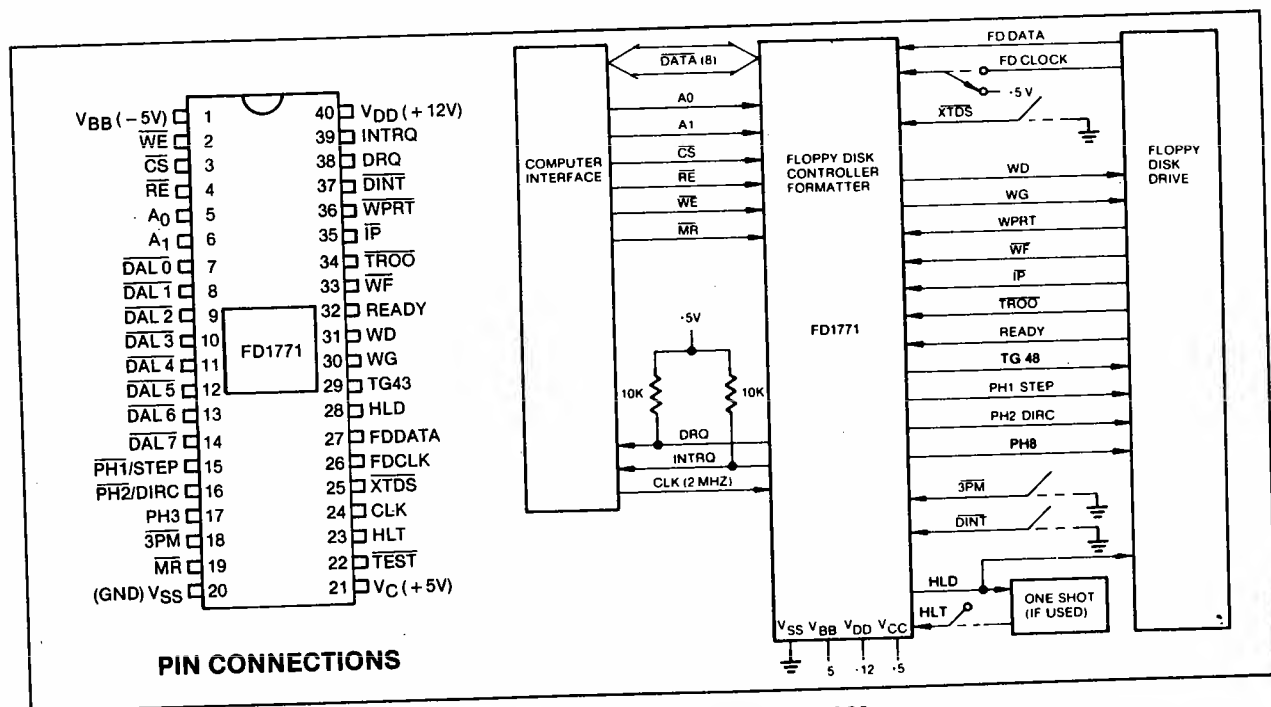
- FLOPPY DISK DRIVE INTERFACE
- SINGLE OR MULTIPLE DRIVE CONTROLLER/FORMATTER
- NEW MINI-FLOPPY CONTROLLER

GENERAL DESCRIPTION

The FD1771 is a MOS/LSI device that performs the functions of a Floppy Disk Controller/Formatter. The device is designed to be included in the disk drive electronics, and contains a flexible interface organization that accommodates the interface signals from most drive manufacturers. The FD1771 is compatible with the IBM 3740 data entry system format.

The processor interface consists of an 8-bit bi-directional bus for data, status, and control word transfers. The FD1771 is set up to operate on a multiplexed bus with other bus-oriented devices.

The FD1771 is fabricated in N-channel Silicon Gate MOS technology and is TTL compatible on all inputs and outputs. The A and B suffixes are for ceramic and plastic packages, respectively.



PIN OUTS

FD1771-01

Pin No.	Pin Name	Symbol	Function																				
1	Power Supplies	V_{BB}/NC	-5V																				
19	MASTER RESET	\overline{MR}	A logic low on this input resets the device and loads "03" into the command register. The Not Ready (Status bit 7) is reset during \overline{MR} ACTIVE. When \overline{MR} is brought to a logic high, a Restore Command is executed, regardless of the state of the Ready signal from the drive.																				
20		V_{SS}	Ground																				
21		V_{CC}	+5V																				
40		V_{DD}	+12V																				
Computer Interface																							
2	WRITE ENABLE	\overline{WE}	A logic low on this input gates data on the DAL into the selected register when \overline{CS} is low.																				
3	CHIP SELECT	\overline{CS}	A logic low on this input selects the chip and enables computer communication with the device.																				
4	READ ENABLE	\overline{RE}	A logic low on this input controls the placement of data from a selected register on the DAL when \overline{CS} is low.																				
5, 6	REGISTER SELECT LINES	A_0, A_1	These inputs select the register to receive/transfer data on the DAL lines under \overline{RE} and \overline{WE} control: <table border="1"> <thead> <tr> <th>A_1</th> <th>A_0</th> <th>\overline{RE}</th> <th>\overline{WE}</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Status Register</td> <td>Command Register</td> </tr> <tr> <td>0</td> <td>1</td> <td>Track Register</td> <td>Track Register</td> </tr> <tr> <td>1</td> <td>0</td> <td>Sector Register</td> <td>Sector Register</td> </tr> <tr> <td>1</td> <td>1</td> <td>Data Register</td> <td>Data Register</td> </tr> </tbody> </table>	A_1	A_0	\overline{RE}	\overline{WE}	0	0	Status Register	Command Register	0	1	Track Register	Track Register	1	0	Sector Register	Sector Register	1	1	Data Register	Data Register
A_1	A_0	\overline{RE}	\overline{WE}																				
0	0	Status Register	Command Register																				
0	1	Track Register	Track Register																				
1	0	Sector Register	Sector Register																				
1	1	Data Register	Data Register																				
7-14	DATA ACCESS LINES	DAL0-DAL7	Eight bit inverted bidirectional bus used for transfer of data, control, and status. This bus is a receiver enabled by \overline{WE} or a transmitter enabled by \overline{RE} .																				
24	CLOCK	CLK	This input requires a free-running 2 MHz $\pm 1\%$ square wave clock for internal timing reference.																				
38	DATA REQUEST	DRQ	This open drain output indicates that the DR contains assembled data in Read operations, or the DR is empty in Write operations. This signal is reset when serviced by the computer through reading or loading the DR in Read or Write operation, respectively. Use 10K pull-up resistor to +5.																				
39	INTERRUPT REQUEST	INTRQ	This open drain output is set at the completion or termination of any operation and is reset when a new command is loaded into the command register. Use 10K pull-up resistor to +5.																				
Floppy Disk Interface:																							
15	Phase 1/Step	$\overline{PH1}/STEP$	If the $\overline{3PM}$ input is a logic low the three-phase motor control is selected and $\overline{PH1}$, $\overline{PH2}$, and $\overline{PH3}$ outputs form a one active low signal out of three. $\overline{PH1}$ is active low after \overline{MR} . If the $\overline{3PM}$ input is a logic high the step and direction motor control is selected. The step output contains a 4 usec high signal for each step and the direction output is active high when stepping in; active low when stepping out.																				
16	Phase 2/Direction	$\overline{PH2}/DIRC$																					
17	Phase 3	$\overline{PH3}$																					
18	3-Phase Motor Select	$\overline{3PM}$																					

Pin No.	Pin Name	Symbol	Function
22	TEST	TEST	This input is used for testing purposes only and should be tied to +5V or left open by the user.
23	HEAD LOAD TIMING	HLT	The HLT input is sampled after 10 ms. When a logic high is sampled on the HLT input the head is assumed to be engaged.
25	EXTERNAL DATA SEPARATION	XTDS	A logic low on this input selects external data separation. A logic high or open selects the internal data separator.
26	FLOPPY DISK CLOCK (External Separation)	FDCLOCK	This input receives the externally separated clock when XTDS = 0. If XTDS = 1, this input should be tied to a logic high.
27	FLOPPY DISK DATA	FDDATA	This input receives the raw read disk data if XTDS=1, or the externally separated data if XTDS=0.
28	HEAD LOAD	HLD	The HLD output controls the loading of the Read-Write head against the media.
29	Track Greater than 43	TG43	This output informs the drive that the Read-Write head is positioned between tracks 44-76. This output is valid only during Read and Write commands.
30	WRITE GATE	WG	This output is made valid when writing is to be performed on the diskette.
31	WRITE DATA	WD	This output contains both clock and data bits of 500 ns duration.
32	Ready	READY	This input indicates disk readiness and is sampled for a logic high before Read or Write commands are performed. If Ready is low, the Read or Write operation is not performed and an interrupt is generated. A Seek operation is performed regardless of the state of Ready. The Ready input appears in inverted format as Status Register bit 7.
33	WRITE FAULT	WF	This input detects wiring faults indications from the drive. When WG=1 and WF goes low, the current Write command is terminated and the Write Fault status bit is set. The WF input should be made inactive (high) when WG becomes inactive.
34	TRACK 00	TR00	This input informs the FD1771 that the Read-Write head is positioned over Track 00 when a logic low.
35	INDEX PULSE	IP	Input, when low for a minimum of 10 usec, informs the FD1771 when an index mark is encountered on the diskette.
36	WRITE PROTECT	WPRT	This input is sampled whenever a Write command is received. A logic low terminates the command and sets the Write Protect status bit.
37	DISK INITIALIZATION	DINT	The input is sampled whenever a Write Track command is received. If DINT=0, the operation is terminated and the Write Protect status bit is set.

ORGANIZATION

The Floppy Disk Formatter block diagram is illustrated on page 4. The primary sections include the parallel processor interface and the Floppy Disk interface.

Data Shift Register: This 8-bit register assembles serial data from the Read Data input (FDDATA) during Read operations and transfers serial data to the Write Data output during Write operations.

Data Register: This 8-bit register is used as a holding register during Disk Read and Write operations. In Disk Read operations the assembled data byte is transferred in parallel to the Data Register from the Data Shift Register. In Disk Write operations information is transferred in parallel from the Data Register to the Data Shift Register.

When executing the Seek command, the Data Register holds the address of the desired Track position. This register can be loaded from the DAL and gated onto the DAL under processor control.

Track Register: This 8-bit register holds the track number of the current Read/Write head position. It is incremented by one every time the head is stepped in (towards track 76) and decremented by one when the head is stepped out (towards track 00). The contents of the register are compared with the recorded track number in the ID field during disk Read, Write, and Verify operations. The Track Register can be

loaded from or transferred to the DAL. This Register should not be loaded when this device is busy.

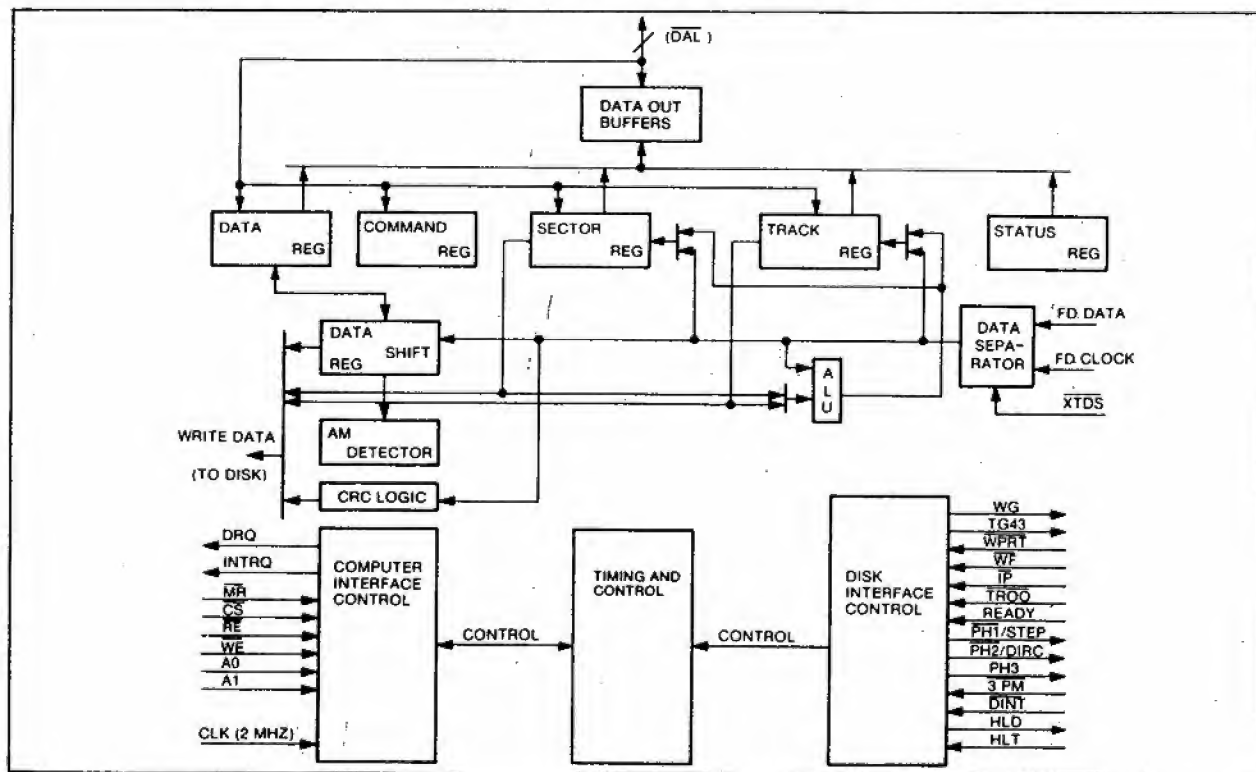
Sector Register (SR): This 8-bit register holds the address of the desired sector position. The contents of the register are compared with the recorded sector number in the ID field during disk Read or Write operations. The Sector Register contents can be loaded from or transferred to the DAL. This register should not be loaded when the device is busy.

Command Register (CR): This 8-bit register holds the command presently being executed. This register should not be loaded when the device is busy unless the execution of the current command is to be overridden. This latter action results in an interrupt. The command register can be loaded from the DAL, but not read onto the DAL.

Status Register (STR): This 8-bit register holds device Status information. The meaning of the Status bits are a function of the contents of the Command Register. This register can be read onto the DAL, but not loaded from the DAL.

CRC Logic: This logic is used to check or to generate the 16-bit Cyclic Redundancy Check (CRC). The polynomial is: $G(x) = x^{16} + x^{12} + x^5 + 1$.

The CRC includes all information starting with the address mark and up to the CRC characters. The CRC register is preset to ones prior to data being shifted through the circuit.



FD1771 BLOCK DIAGRAM

Arithmetic/Logic Unit (ALU): The ALU is a serial comparator, incrementer, and decrements and is used for register modification and comparisons with the disk recorded ID field.

AM Detector: The Address Mark detector is used to detect ID, Data, and Index address marks during Read and Write operations.

Timing and Control: All computer and Floppy Disk Interface controls are generated through this logic. The internal device timing is generated from a 2.0 MHz external crystal clock.

PROCESSOR INTERFACE

The interface to the processor is accomplished through the eight Data Access Lines (DAL) and associated control signals. The DAL are used to transfer Data, Status, and Control words out of, or into the FD1771. The DAL are three-state buffers that are enabled as output drivers when Chip Select (CS) and Read Enable (RE) are active (low logic state) or act as input receivers when CS and Write Enable (WE) are active.

When transfer of data with the Floppy Disk Controller is required by the host processor, the device address is decoded and CS is made low. The least-significant address bits A1 and A0, combined with the signals RE during a Read operation or WE during a Write operation are interpreted as selecting the following registers:

A1-A0	READ (RE)	WRITE (WE)
0 0	Status Register	Command Register
0 1	Track Register	Track Register
1 0	Sector Register	Sector Register
1 1	Data Register	Data Register

During Direct Memory Access (DMA) types of data transfers between the Data Register of the FD1771 and the Processor, the Data Request (DRQ) output is used in Data Transfer control. This signal also appears as status bit 1 during Read and Write operations.

On Disk Read operations the Data Request is activated (set high) when an assembled serial input byte is transferred in parallel to the Data Register. This bit is cleared when the Data Register is read by the processor. If the Data Register is read after one or more characters are lost, by having new data transferred into the register prior to processor readout, the Lost Data bit is set in the Status Register. The Read operation continues until the end of sector is reached.

On Disk Write operations the Data Request is activated when the Data Register transfers its contents to the Data Shift Register, and requires a new data byte. It is reset when the Data Register is loaded with new data by the processor. If new data is not loaded

at the time the next serial byte is required by the Floppy Disk, a byte of zeroes is written on the diskette and the Lost Data bit is set in the Status Register.

The Lost Data bit and certain other bits in the Status Register will activate the interrupt request (INTRQ). The interrupt line is also activated with normal completion or abnormal termination of all controller operations. The INTRQ signal remains active until reset by reading the Status Register to the processor or by the loading of the Command Register. In addition, the INTRQ is generated if a Force Interrupt command condition is met.

FLOPPY DISK INTERFACE

The Floppy Disk interface consists of head positioning controls, write gate controls, and data transfers. A 2.0 MHz \pm 1% square wave clock is required at the CLK input for internal control timing (may be 1.0 MHz for mini floppy).

HEAD POSITIONING

Four commands cause positioning of the Read-Write head (see Command Section). The period of each positioning step is specified by the r field in bits 1 and 0 of the command word. After the last directional step, an additional 10 milliseconds of head settling time takes place. The four programmable stepping rates are tabulated below.

The rates (shown in Table 1) can be applied to a Three-Phase Motor or a Step-Direction Motor through the device interface. When the 3PM input is connected to ground, the device operates with a three-phase motor control interface, with one active low signal per phase on the three output signals PH1, PH2, and PH3. The stepping sequence, when stepping in, is Phases 1-2-3-1, and when stepping out, Phases 1-3-2-1. Phase 1 is active low after Master Reset. Note: PH3 needs an inverter if used.

The Step-Direction Motor Control interface is activated by leaving input 3PM open or connecting it to +5V. The Phase 1 pin PH1 becomes a Step pulse of 4 microseconds width. The Phase 2 pin PH2 becomes a direction control with a high voltage on this pin indicating a Step In, and a low voltage indicating a Step Out. The Direction output is valid a minimum of 24 μ s prior to the activation of the Step pulse.

When a Seek, Step or Restore command is executed, an optional verification of Read-Write head position can be performed by setting bit 2 in the command word to a logic 1. The verification operation begins at the end of the 10 millisecond settling time after the head is loaded against the media. The track number from the first encountered ID Field is compared against the contents of the Track Register. If the track numbers compare and the ID Field Cyclic Redundancy Check (CRC) is correct, the verify operation is complete. If track comparison is not

made but the CRC checks, an interrupt is generated, the Seek Error status (Bit 4) is set and the Busy status bit is reset.

Table 1. STEPPING RATES

r ₁ r ₀	1771-X1 CLK = 2 MHz TEST = 1	1771-X1 CLK = 1 MHz TEST = 1	1771 or -X1 CLK = 2 MHz TEST = 0	1771 or -X1 CLK = 1 MHz TEST = 0
0 0	6ms	12ms		
0 1	6ms	12ms	Approx. 400μs*	Approx. 800μs*
1 0	10ms	20ms		
1 1	20ms	40ms		

* For exact times consult WDC.

The Head Load (HLD) output controls the movement of the read/write head against the disk for data recording or retrieval. It is activated at the beginning of a Read, Write (E Flag On) or Verify operation, or a Seek or Step operation with the head load bit, h, a logic one remains activated until the third index pulse following the last operation which uses the read/write head. Reading or Writing does not occur until a minimum of 10 msec delay after the HLD signal is made active. If executing the type 2 commands with the E flag off, there is no 10 msec delay and the head is assumed to be engaged. The delay is determined by sampling of the Head Load Timing (HLT) input after 10 msec. A high state input, generated from the Head Load output transition and delayed externally, identifies engagement of the head against the disk. In the Seek and Step commands, the head is loaded at the start of the command execution when the h bit is a logic one. In a verify command the head is loaded after stepping to the destination track on the disk whenever the h bit is a logic zero.

DISK READ OPERATION

The 2.0 MHz external clock provided to the device is internally divided by 4 to form the 500 kHz clock rate for data transfer. When reading data from a diskette this divider is synchronized to transitions of the Read Data (FDDATA) input. When a transition does not occur on the 500 kHz clock active state, the clock divider circuit injects a clock to maintain a continuous 500 kHz data clock. The 500 kHz data clock is further divided by 2 internally to separate the clock and information bits. The divider is phased to the information by the detection of the address mark.

In the internal data read and separation mode the Read Data input toggles from one state to the opposite state for each logic one bit of clock or information. This signal can be derived from the amplified, differentiated, and sliced Read Head signal, or by the output of a flip-flop toggling on the Read Data pulses. This input is sampled by the 2 MHz clock to detect transitions.

The chip can also operate on externally separated

data, as supplied by methods such as Phase Lock loop, One Shots, or variable frequency oscillators. This is accomplished by grounding the External Data Separator (XTDS) INPUT. When the Read Data input makes a high-to-low transition, the information input to the FDDATA line is clocked into the Data Shift Register. The assembled 8-bit data from the Data Shift Register are then transferred to the Data Register.

The normal sector length for read or Write operations with the IBM 3740 format is 128 bytes. This format or binary multiples of 128 bytes will be adopted by setting a logic 1 in Bit 3 of the Read and Write commands. Additionally, a variable sector length feature is provided which allows an indicator recorded in the ID Field to control the length of the sector. Variable sector lengths can be read or written in Read or Write commands, respectively, by setting a logic 0 in Bit 3 of the command word. The sector length indicator specifies the number of 16 byte groups or 16 x N, where N is equal to 1 to 256 groups. An indicator of all zeroes is interpreted as 256 sixteen byte groups.

DISK WRITE OPERATION

After data is loaded from the processor into the Data Register, and is transferred to the Data Shift Register, data will be shifted serially through the Write Data (WD) output. Interlaced with each bit of data is a positive clock pulse of 0.5 μsec duration. This signal may be used to externally toggle a flip-flop to control the direction of Write Current flow.

When writing is to take place on the diskette the Write Gate (WG) output is activated, allowing current to flow into the Read/Write head. As a precaution to erroneous writing, the first data byte must be loaded into the Data Register in response to a Data Request from the FD1771 before the Write Gate signal can be activated.

Writing is inhibited when the Write Protect input is a logic low, in which case any Write command is immediately terminated, an interrupt is generated and the Write Protect status bit is set. The Write Fault input, when activated, signifies a writing fault condition detected in disk drive electronics such as failure to detect write current flow when the Write Gate is activated. On detection of this fault the FD1771 terminates the current command, and sets the Write Fault bit (bit 5) in the Status Word. The Write Fault input should be made inactive when the Write Gate output becomes inactive.

Whenever a Read or Write command is received the FD1771 samples the READY input. If this input is logic low the command is not executed and an interrupt is generated. The Seek or Step commands are performed regardless of the state of the READY input.

COMMAND DESCRIPTION

The FD1771 will accept and execute eleven commands. Command words should only be loaded in the Command Register when the Busy status bit is off (status bit 0). The one exception is the Force Interrupt command. Whenever a command is being executed, the Busy status bit is set. When a command is completed, an interrupt is generated and the Busy status bit is reset. The Status Register indicates whether the completed command encountered an error or was fault-free. For ease of discussion, commands are divided into four types. Commands and types are summarized in Table 2.

TYPE 1 COMMANDS

The Type 1 Commands include the RESTORE, SEEK, STEP, STEP-IN, and STEP-OUT commands. Each of the Type 1 Commands contain a rate field (r_0r_1), which determines the stepping motor rate as defined in Table 1, page 4.

The Type 1 Commands contain a head load flag (h) which determines if the head is to be loaded at the beginning of the command. If $h=1$, the head is loaded at the beginning of the command (HLD output is made active). If $h=0$, HLD is deactivated.

Table 2. COMMAND SUMMARY

TYPE	COMMAND	BITS							
		7	6	5	4	3	2	1	0
I	Restore	0	0	0	0	h	V	r_1	r_0
I	Seek	0	0	0	1	h	V	r_1	r_0
I	Step	0	0	1	u	h	V	r_1	r_0
I	Step In	0	1	0	u	h	V	r_1	r_0
I	Step Out	0	1	1	u	h	V	r_1	r_0
II	Read Command	1	0	0	m	b	E	0	0
II	Write Command	1	0	1	m	b	E	a_1a_0	
III	Read Address	1	1	0	0	0	E	0	0
III	Read Track	1	1	1	0	0	1	0	\bar{s}
III	Write Track	1	1	1	1	0	1	0	0
IV	Force Interrupt	1	1	0	1	l_3	l_2	l_1	l_4

Note: Bits shown in TRUE form.

Table 3. FLAG SUMMARY

TYPE I	
h = Head Load flag (Bit 3)	
$h = 1$, Load head at beginning	
$h = 0$, Do not load head at beginning	
V = Verify flag (Bit 2)	
$V = 1$, Verify on last track	
$V = 0$, No verify	
r_1r_0 = Stepping motor rate (Bits 1-0)	
Refer to Table 1 for rate summary	
u = Update flag (Bit 4)	
$u = 1$, Update Track register	
$u = 0$, No update	

Table 4. FLAG SUMMARY

TYPE II	
m = Multiple Record flag (Bit 4)	
$m = 0$, Single Record	
$m = 1$, Multiple Records	
b = Block length flag (Bit 3)	
$b = 1$, IBM format (128 to 1024 bytes)	
$b = 0$, Non-IBM format (16 to 4096 bytes)	
a_1a_0 = Data Address Mark (Bits 1-0)	
$a_1a_0 = 00$, FB (Data Mark)	
$a_1a_0 = 01$, FA (User defined)	
$a_1a_0 = 10$, F9 (User defined)	
$a_1a_0 = 11$, F8 (Deleted Data Mark)	

Table 5. FLAG SUMMARY

TYPE III	
s = Synchronize flag (Bit 0)	
$\bar{s} = 0$, Synchronize to AM	
$\bar{s} = 1$, Do Not Synchronize to AM	
TYPE IV	
li = Interrupt Condition flags (Bits 3-0)	
$l_0 = 1$, Not Ready to Ready Transition	
$l_1 = 1$, Ready to Not Ready Transition	
$l_2 = 1$, Index Pulse	
$l_3 = 1$, Immediate interrupt	
E = Enable HLD and 10 msec Delay	
$E = 1$, Enable HLD, HLT and 10 msec Delay	
$E = 0$, Head is assumed Engaged and there is no 10 msec Delay	

Once the head is loaded, the head will remain engaged until the FD1771 receives a command that specifically disengages the head. If the FD1771 does not receive any commands after two revolutions of the disk, the head will be automatically disengaged (HLD made inactive). The Head Load Timing Input is sampled after a 10 ms delay, when reading or writing on the disk is to occur.

The Type 1 Commands also contain a verification (V) flag which determines if a verification operation is to take place on the destination track. If $V=1$, a verification is performed; if $V=0$, no verification is performed.

During verification, the head is loaded and after an internal 10 ms delay, the HLT input is sampled. When HLT is active (logic true), the first encountered ID field is read off the disk. The track address of the ID Field is then compared to the Track Register; if there is a match and a valid ID CRC, the verification is complete, an interrupt is generated and the BUSY status bit is reset. If there is not a match but there is

FD1771-01

FD1771-01

valid ID CRC, an interrupt is generated, the Seek Error status bit (Status Bit 4) is set and the BUSY status bit is reset. If there is a match but not a valid CRC, the CRC error status bit is set (Status Bit 3), and the next encountered ID Field is read from the disk for the verification operation. If an ID Field with a valid CRC cannot be found after two revolutions of the disk, the FD1771 terminates the operation and sends an interrupt (INTRQ).

The STEP, STEP-IN, and STEP-OUT commands contain an UPDATE flag (U). When U=1, the track register is updated by one for each step. When U=0, the track register is not updated.

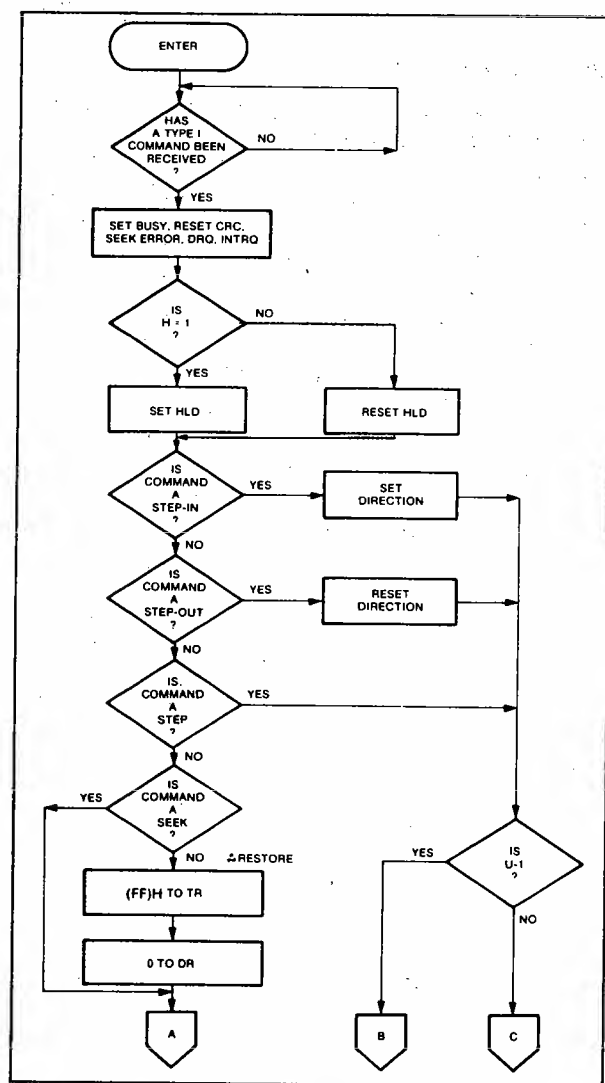
RESTORE (SEEK TRACK 0)

Upon receipt of this command the Track 00 ($\overline{TR00}$) input is sampled. If $\overline{TR00}$ is active low indicating the Read-Write head is positioned over track 0, the Track

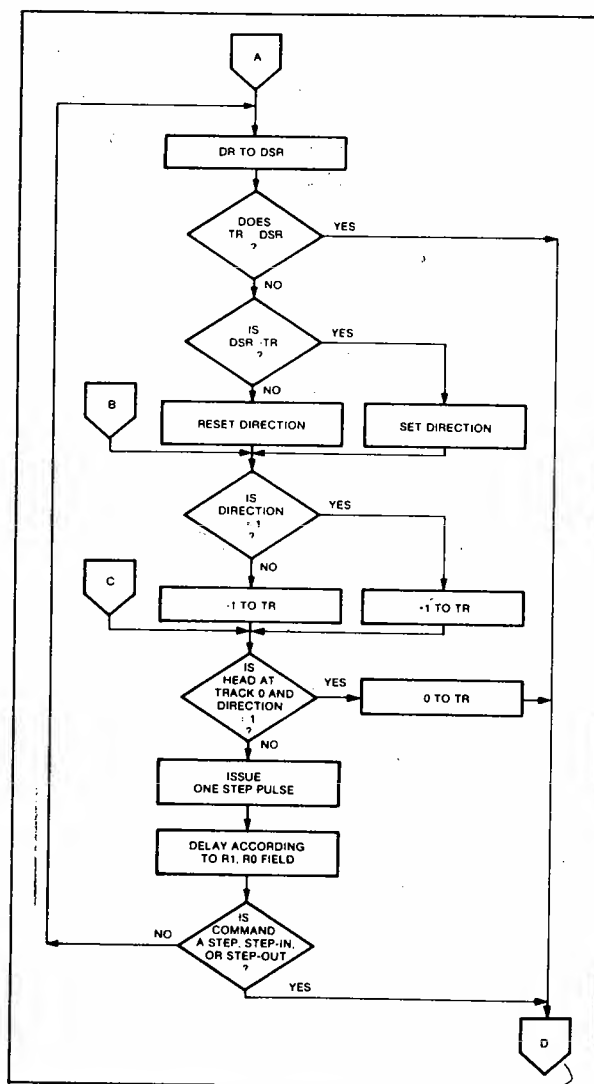
Register is loaded with zeroes and an interrupt is generated. If $\overline{TR00}$ is not active low, stepping pulses (pins 15 to 17) at a rate specified by the r_1r_0 field are issued until the $\overline{TR00}$ input is activated. At this time the TR is loaded with zeroes and an interrupt is generated. If the $\overline{TR00}$ input does not go active low after 255 stepping pulses, the FD1771 terminates operation, interrupts, and sets the Seek error status bit. Note that the RESTORE command is executed when \overline{MR} goes from an active to an inactive state. A verification operation takes place if the V flag is set. The h bit allows the head to be loaded at the start of command.

SEEK

This command assumes that the Track Register contains the track number of the current position of the Read-Write head and the Data Register contains the desired track number. The FD1771 will update the



TYPE I COMMAND FLOW



TYPE I COMMAND FLOW

Track register and issue stepping pulses in the appropriate direction until the contents of the Track register are equal to the contents of the data register (the desired track location). A verification operation takes place if the V flag is on. The h bit allows the head to be loaded at the start of the command. An interrupt is generated at the completion of the command.

STEP

Upon receipt of this command, the FD1771 issues one stepping pulse to the disk drive. The stepping motor direction is the same as in the previous step command. After a delay determined by the r_{1r0} field, a verification takes place if the V flag is on. If the u flag is on, the TR is updated. The h bit allows the head to be loaded at the start of the command. An

interrupt is generated at the completion of the command.

STEP-IN

Upon receipt of this command, the FD1771 issues one stepping pulse in the direction towards track 76.^{*} If the u flag is on, the Track Register is incremented by one. After a delay determined by the r_{1r0} field, a verification takes place if the V flag is on. The h bit allows the head to be loaded at the start of the command. An interrupt is generated at the completion of the command.

STEP-OUT

Upon receipt of this command, the FD1771 issues one stepping pulse in the direction towards track 0. If the u flag is on, the TR is decremented by one. After a delay determined by the r_{1r0} field, a verification takes place if the V flag is on. The h bit allows the head to be loaded at the start of the command. An interrupt is generated at the completion of the command.

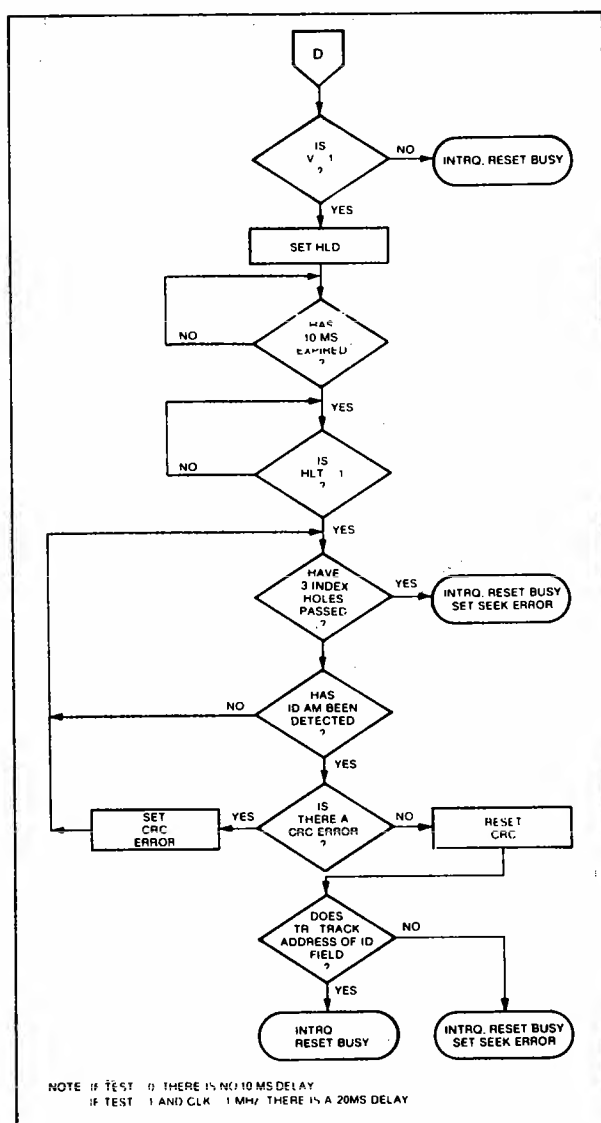
TYPE II COMMANDS

The Type II Commands include the Read Sector(s) and Write Sector(s) commands. Prior to loading the Type II command into the COMMAND REGISTER, the computer must load the Sector Register with the desired sector number. Upon receipt of the Type II command, the Busy status bit is set. If the E flag=1 (this is the normal case), HLD is made active and HLT is sampled after a 10 msec delay. If the E flag is 0, the head is assumed to be engaged and there is no 10 msec delay. The ID field and the Data Field format are shown below.

When an ID field is located on the disk, the FD1771 compares the track number of the ID field with the Track Register. If there is not a match, the next encountered ID field is read and a comparison is again made. If there was a match, the Sector Number of the ID field is compared with the Sector Register. If there is not a Sector match, the next encountered ID field is read off the disk and comparisons again made. If the ID field CRC is correct, the data field is then located and will be either written into, or read from depending on the command. The FD1771 must find an ID field with a track number, Sector number, and CRC within two revolutions of the disk; otherwise, the Record Not Found status bit is set (Status bit 3) and the command is terminated with an interrupt.

Each of the Type II Commands contain a (b) flag which in conjunction with the sector length field contents of the ID determines the length (number of characters) of the Data field.

For IBM 3740 compatibility, the b flag should equal 1. The numbers of bytes in the data field (sector) is then 128×2^n where $n = 0, 1, 2, 3$.



TYPE I COMMAND FLOW

GAP	ID AM	TRACK NUMBER	ZERO	SECTOR NUMBER	SECTOR LENGTH	CRC 1	CRC 2	GAP	DATA AM	DATA FIELD	CRC 1	CRC 2
ID FIELD									DATA FIELD			

IDAM = ID Address Mark — DATA = (FE)₁₆ CLK = (C7)₁₆

Data AM = Data Address Mark — DATA = (F8, F9, FA, or FB), CLK = (C7)₁₆

For b = 1

Sector Length Field (Hex)	Number of Bytes in Sector (Decimal)
00	128
01	256
02	512
03	1024

When the b flag equals zero, the sector length field (n) multiplied by 16 determines the number of bytes in the sector or data field as shown below.

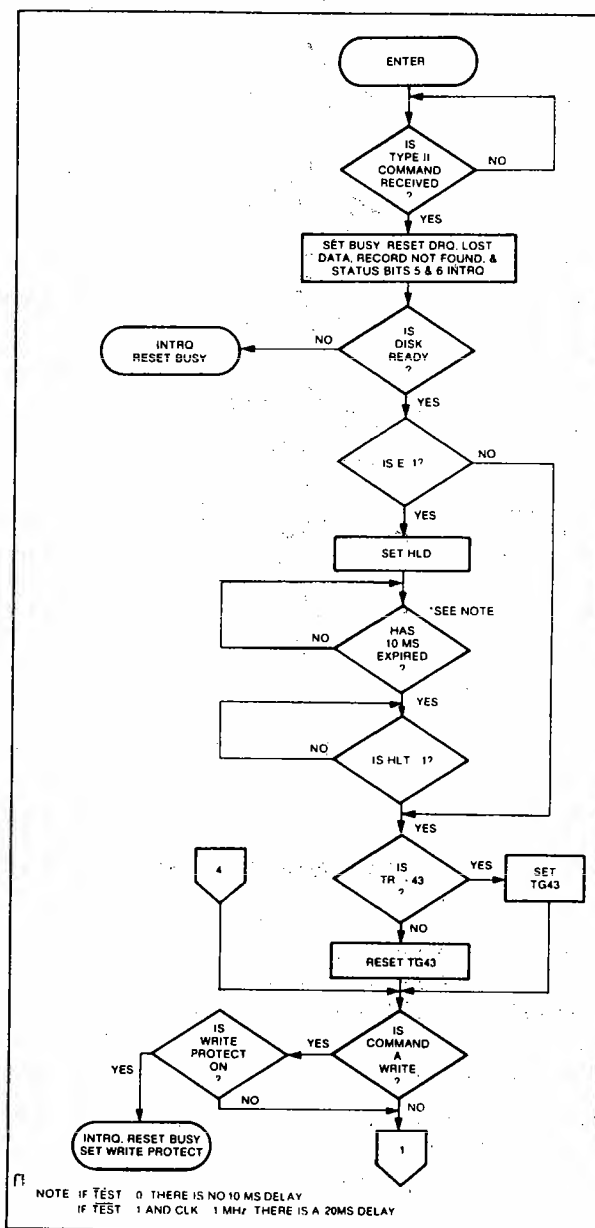
For b = 0

Sector Length Field (Hex)	Number of Bytes in Sector (Decimal)
01	16
02	32
03	48
04	64
•	•
•	•
•	•
FF	4080
00	4096

Each of the Type II commands also contain a (m) flag which determines if the multiple records (sectors) are to be read or written, depending upon the command. If m=0 a single sector is read or written and an interrupt is generated at the completion of the command. If m=1, multiple records are read or written with the sector register internally updated so that an address verification can occur on the next record. The FD1771 will continue to read or write multiple records and update the sector register until the sector register exceeds the number of sectors on the track or until the Force Interrupt command is loaded into the command register, which terminated the command and generates an interrupt.

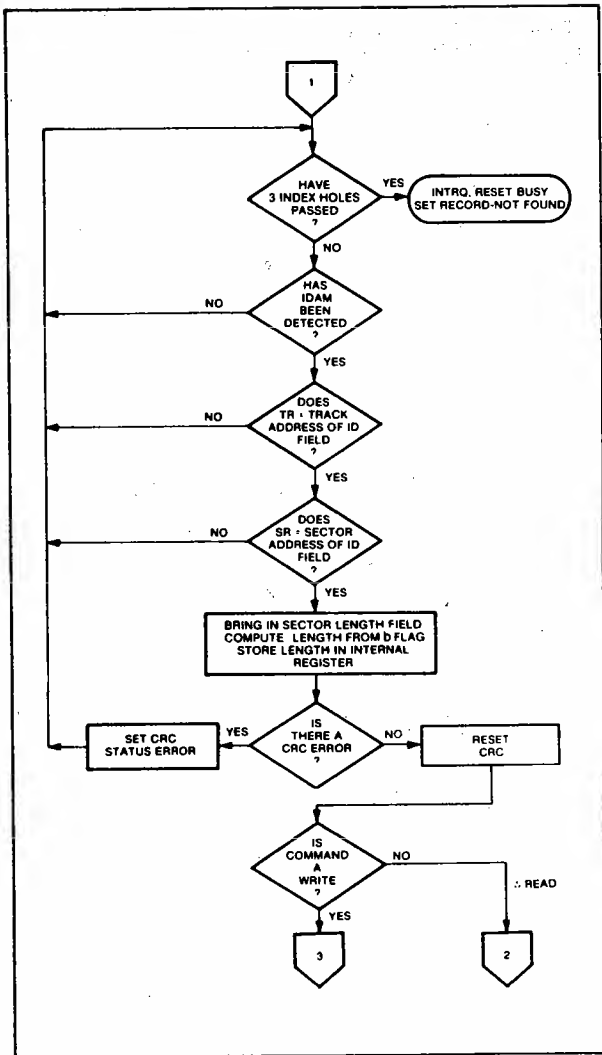
READ COMMAND

Upon receipt of the Read command, the head is loaded, the BUSY status bit set, and when an ID field is encountered that has the correct track number, correct sector number, and correct CRC, the data field is presented to the computer. The Data Address Mark of the data field must be found within 28 bytes of the correct field; if not, the Record Not Found status bit is set and the operation is terminated. When the first character or byte of the data field has been



TYPE II COMMAND FLOW

shifted through the DSR, it is transferred to the DR, and DRQ is generated. When the next byte is accumulated in the DSR, it is transferred to the DR and another DRQ is generated. If the computer has not read the previous contents of the DR before a new character is transferred that character is lost and the

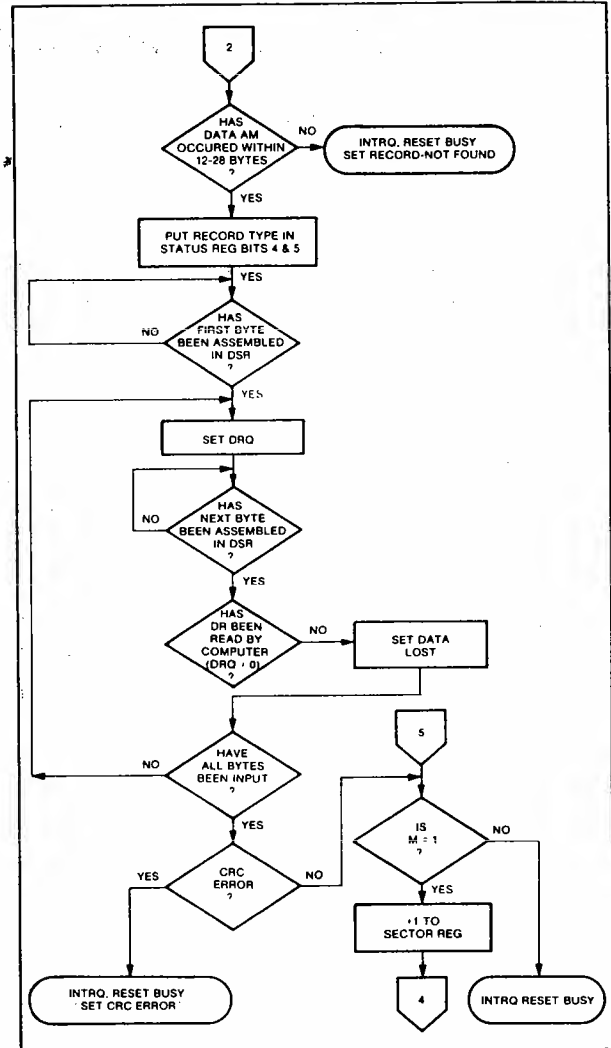


TYPE II COMMAND FLOW

Lost Data status bit is set. This sequence continues until the complete data field has been input to the computer. If there is a CRC error at the end of the data field, the CRC error status bit is set, and the command is terminated (even if it is a multiple record command).

At the end of the Read operation, the type of Data Address Mark encountered in the data field is recorded in the Status Register (Bits 5 and 6) as shown below.

Status Bit 6	Status Bit 5	Data AM (Hex)
0	0	FB
0	1	FA
1	0	F9
1	1	F8



TYPE II COMMAND FLOW

WRITE COMMAND

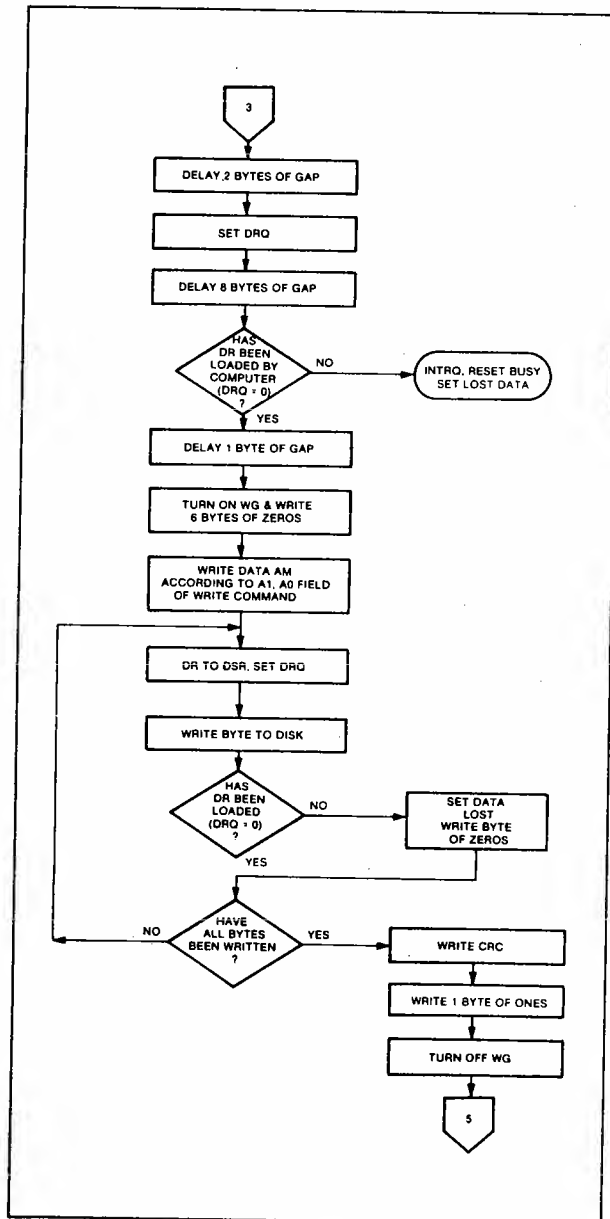
Upon receipt of the Write command, the head is loaded (HLD active) and the BUSY status bit is set. When an ID field is encountered that has the correct track number, correct sector number, and correct CRC, a DRQ is generated. The FD1771 counts off 11 bytes from the CRC field and the Write Gate (WG) output is made active if the DRQ is serviced (i.e., the DR has been loaded by the computer). If DRQ has not been serviced, the command is terminated and the Lost Data status bit is set. If the DRQ has been serviced, the WG is made active and six bytes of zeros are then written on the disk. At this time the Data Address Mark is then written on the disk as determined by the $a_1 a_0$ field of the command as shown on next page.

The FD1771 then writes the data field and generates DRQs to the computer. If the DRQ is not serviced in

FD1771-01

a ₁	a ₀	Data Mark (Hex)	Clock Mark (Hex)
0	0	FB	C7
0	1	FA	C7
1	0	F9	C7
1	1	F8	C7

time for continuous writing the Lost Data status bit is set and a byte of zeros is written on the disk. The command is not terminated. After the last data byte has been written on the disk, the two-byte CRC is computed internally and written on the disk followed by one byte gap of logic ones. The WG output is then deactivated.



TYPE II COMMAND FLOW

TYPE III COMMANDS

READ Address

Upon receipt of the Read Address command, the head is loaded and the BUSY Status bit is set. The next encountered ID field is then read in from the disk, and the six data bytes of the ID field are assembled and transferred to the DR, and a DRQ is generated for each byte. The six bytes of the ID field are shown below.

TRACK ADDR	SIDE NUMBER	SECTOR ADDRESS	SECTOR LENGTH	CRC 1	CRC 2
1	2	3	4	5	6

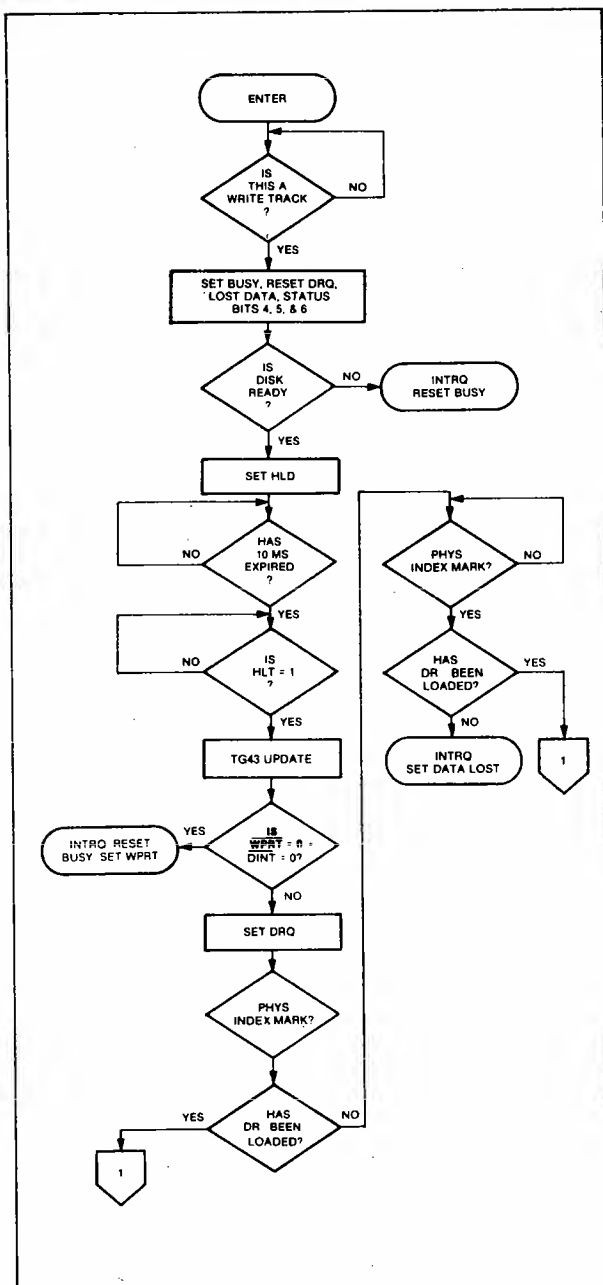
Although the CRC characters are transferred to the computer, the FD1771 checks for validity and the CRC error status bit is set if there is a CRC error. The Sector Address of the ID field is written into the Sector Register. At the end of the operation an interrupt is generated and the BUSY Status is reset.

READ TRACK

Upon receipt of the Read Track command, the head is loaded and the BUSY status bit is set. Reading starts with the leading edge of the first encountered index mark and continues until the next index pulse. As each byte is assembled it is transferred to the Data Register and the Data Request is generated for each byte. No CRC checking is performed. Gaps are included in the input data stream. If bit 0(S) of the command is a 0, the accumulation of bytes is synchronized to each Address Mark encountered. Upon completion of the command, the interrupt is activated.

WRITE TRACK

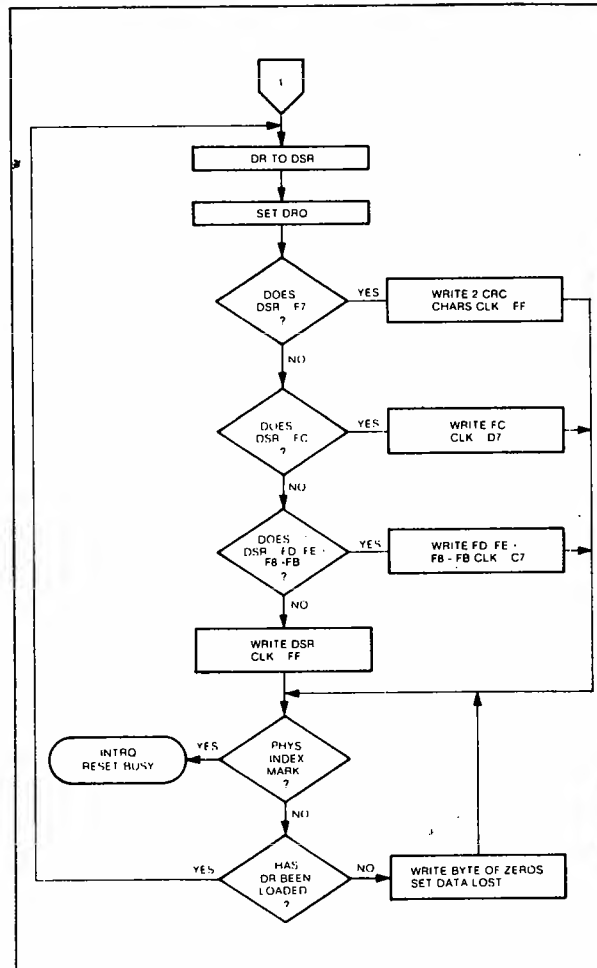
Upon receipt of the Write Track command, the head is loaded and the BUSY status bit is set. Writing starts with the leading edge of the first encountered index pulse and continues until the next index pulse, at which time the interrupt is activated. The Data Request is activated immediately upon receiving the command, but writing will not start until after the first byte has been loaded into the Data Register. If the DR has not been loaded by the time the index pulse is encountered the operation is terminated making the device Not Busy, the Lost Data status bit is set, and the Interrupt is activated. If a byte is not present in the DR when needed, a byte of zeros is substituted. Address Marks and CRC characters are written on the disk by detecting certain data byte patterns in the outgoing data stream as shown in the table below. The CRC generator is initialized when any data byte from F8 to FE is about to be transferred from the DR to the DSR.



TYPE III COMMAND WRITE TRACK

CONTROL BYTES FOR INITIALIZATION

DATA PATTERN (HEX)	INTERPRETATION	CLOCK MARK (HEX)
F7	Write CRC Character	FF
F8	Data Address Mark	C7
F9	Data Address Mark	C7
FA	Data Address Mark	C7
FB	Data Address Mark	C7
FC	Index Address Mark	D7
FD	Spare	
FE	ID Address Mark	C7



TYPE III COMMAND WRITE TRACK

The Write Track Command will not execute if the DINT input is grounded; instead, the Write Protect status bit is set and the interrupt is activated. Note that one F7 pattern generates two CRC characters.

TYPE IV COMMAND

Force Interrupt

This command can be loaded into the command register at any time. If there is a current command under execution (BUSY status bit set), the command will be terminated and an interrupt will be generated when the condition specified in the I₀ through I₃ field is detected. The interrupt conditions are shown below:

- I₀ = Not-Ready-To-Ready Transition
- I₁ = Ready-To-Not-Ready Transition
- I₂ = Every Index Pulse
- I₃ = Immediate Interrupt (Requires reset, see Note)

NOTE: If I₀ - I₃ = 0, there is no interrupt generated but the current command is terminated and busy is reset. This is the only command that will clear the immediate interrupt.

FD1771-01

STATUS DESCRIPTION

Upon receipt of any command, except the Force Interrupt command, the Busy Status bit is set and the rest of the status bits are updated or cleared for the new command. If the Force Interrupt Command is received when there is a current command under execution, the Busy status bit is reset, and the rest of the status bits are unchanged. If the Force Interrupt command is received when there is not a current command under execution, the Busy Status bit is

reset and the rest of the status bits are updated or cleared. In this case, Status reflects the Type I commands.

The format of the Status Register is shown below.

(BITS)							
7	6	5	4	3	2	1	0
S7	S6	S5	S4	S3	S2	S1	S0

Status varies according to the type of command executed as shown in Table 6.

Table 6. STATUS REGISTER SUMMARY

BIT	ALL TYPE I COMMANDS	READ ADDRESS	READ	READ TRACK	WRITE	WRITE TRACK
S7	NOT READY	NOT READY	NOT READY	NOT READY	NOT READY	NOT READY
S6	WRITE PROTECT	0	RECORD TYPE	0	WRITE PROTECT	WRITE PROTECT
S5	HEAD ENGAGED	0	RECORD TYPE	0	WRITE FAULT	WRITE FAULT
S4	SEEK ERROR	ID NOT FOUND	RECORD NOT FOUND	0	RECORD NOT FOUND	0
S3	CRC ERROR	CRC ERROR	CRC ERROR	0	CRC ERROR	0
S2	TRACK 0	LOST DATA	LOST DATA	LOST DATA	LOST DATA	LOST DATA
S1	INDEX	DRQ	DRQ	DRQ	DRQ	DRQ
S0	BUSY	BUSY	BUSY	BUSY	BUSY	BUSY

STATUS FOR TYPE I COMMANDS

BIT	NAME	MEANING
S7	NOT READY	This bit when set indicates the drive is not ready. When reset it indicates that the drive is ready. This bit is an inverted copy of the READY input and logically "ored" with MR.
S6	PROTECTED	When set, indicates Write Protect is activated. This bit is an inverted copy of WRPT input.
S5	HEAD LOADED	When set, it indicates the head is loaded and engaged. This bit is a logical "and" of HLD and HLT signals.
S4	SEEK ERROR	When set, the desired track was not verified. This bit is reset to 0 when updated.
S3	CRC ERROR	When set, there was one or more CRC errors encountered on an unsuccessful track verification operation. This bit is reset to 0 when updated.
S2	TRACK 00	When set, indicates Read-Write head is positioned to Track 0. This bit is an inverted copy of the TR00 input.
S1	INDEX	When set, indicates index mark detected from drive. This bit is an inverted copy of the IP input.
S0	BUSY	When set, command is in progress. When reset, no command is in progress.

STATUS BITS FOR TYPE II AND III COMMANDS

BIT	NAME	MEANING
S7	NOT READY	This bit when set indicates the drive is not ready. When reset, it indicates that the drive is ready. This bit is an inverted copy of the READY input and "ored" with MR. The TYPE II and III Commands will not execute unless the drive is ready.
S6	RECORD TYPE/ WRITE PROTECT	On Read Record: It indicates the MSB of record-type code from data field address mark. On Read Track: Not Used. On any Write Track: It indicates a Write Protect. This bit is reset when updated.
S5	RECORD TYPE/WRITE FAULT	On Read Record: It indicates the LSB of record-type code from data field address mark. On Read Track: Not Used. On any Write Track: It indicates a Write Fault. This bit is reset when updated.
S4	RECORD NOT FOUND	When set, it indicates that the desired track and sector were not found. This bit is reset when updated.
S3	CRC ERROR	If S4 is set, an error is found in one or more ID fields; otherwise it indicates error in data field. This bit is reset when updated.
S2	LOST DATA	When set, it indicates the computer did not respond to DRQ in one byte time. This bit is reset to zero when updated.
S1	DATA REQUEST	This bit is a copy of the DRQ output. When set, it indicates the DR is full on a Ready operation or the DR is empty on a Write operation. This bit is reset to zero when updated.
S0	BUSY	When set, command is under execution. When reset, no command is under execution.

FORMATTING THE DISK (Refer to section on Type III Commands for flow diagrams.)

Formatting the disk is a relatively simple task when operating programmed I/O or when operating under DMA control with a large amount of memory. When operating under DMA with limited amount of memory, formatting is a more difficult task. This is because gaps as well as data must be provided at the computer interface.

Formatting the disk is accomplished by positioning the R/W head over the desired track number and issuing the Write Track command. Upon receipt of the Write Track command, the FD1771 raises the Data Request signal. At this point in time, the user loads the Data Register with desired data to be written on the disk. For every byte of information to be written on the disk, a Data Request is generated. This sequence continues from one index mark to the next index mark. Normally, whatever data pattern appears in the Data Register is written on the disk with a clock mark of (FF)₁₆. However, if the FD1771 detects a data pattern on F7 through FE in the Data Register, this is interpreted as data address marks with missing clocks or CRC generation. For

instance, an FE pattern will be interpreted as an ID address mark (DATA-FE, CLK-C7) and the CRC will be initialized. An F7 pattern will generate two CRC characters. As a consequence, the patterns F7 through FE must not appear in the gaps, data fields, or ID fields. Also, CRCs must be generated by an F7 pattern.

Disks may be formatted in IBM 3740 formats with sector lengths of 128,256,512, or 1024 bytes, or may be formatted in non-IBM format with sector lengths of 16 to 4096 bytes in 16-byte increments. IBM 3740 at the present time only defines two formats. One format with 128 bytes/sector and the other with 256 bytes/sector. The next section deals with the IBM 3740 format with 128 bytes/sector followed by a section of non-IBM formats.

IBM 3740 Formats — 128 Bytes/Sector

The IBM format with 128 bytes/sector is depicted in the Track Format figure on the following page. In order to create this format, the user must issue the Write Track command, and load the data register with the following values. For every byte to be written, there is one data request.

FD1771-01

Number of Bytes	Hex Value of Byte Written
40	00 or FF
6	00
1	FC (Index Mark)
* 26	00 or FF
6	00
1	FE (ID Address Mark)
1	Track Number (0 through 4C)
1	00
1	Sector Number (1 through 1A)
1	00
1	F7 (two CRCs written)
11	00 or FF
6	00
1	FB (Data Address Mark)
128	Data (IBM uses E5)
1	F7 (two CRCs written)
27	00 or FF
247 **	00 or FF

*Write bracketed field 26 times.

**Continue writing until FD1771 interrupts out. Approximately 247 bytes.

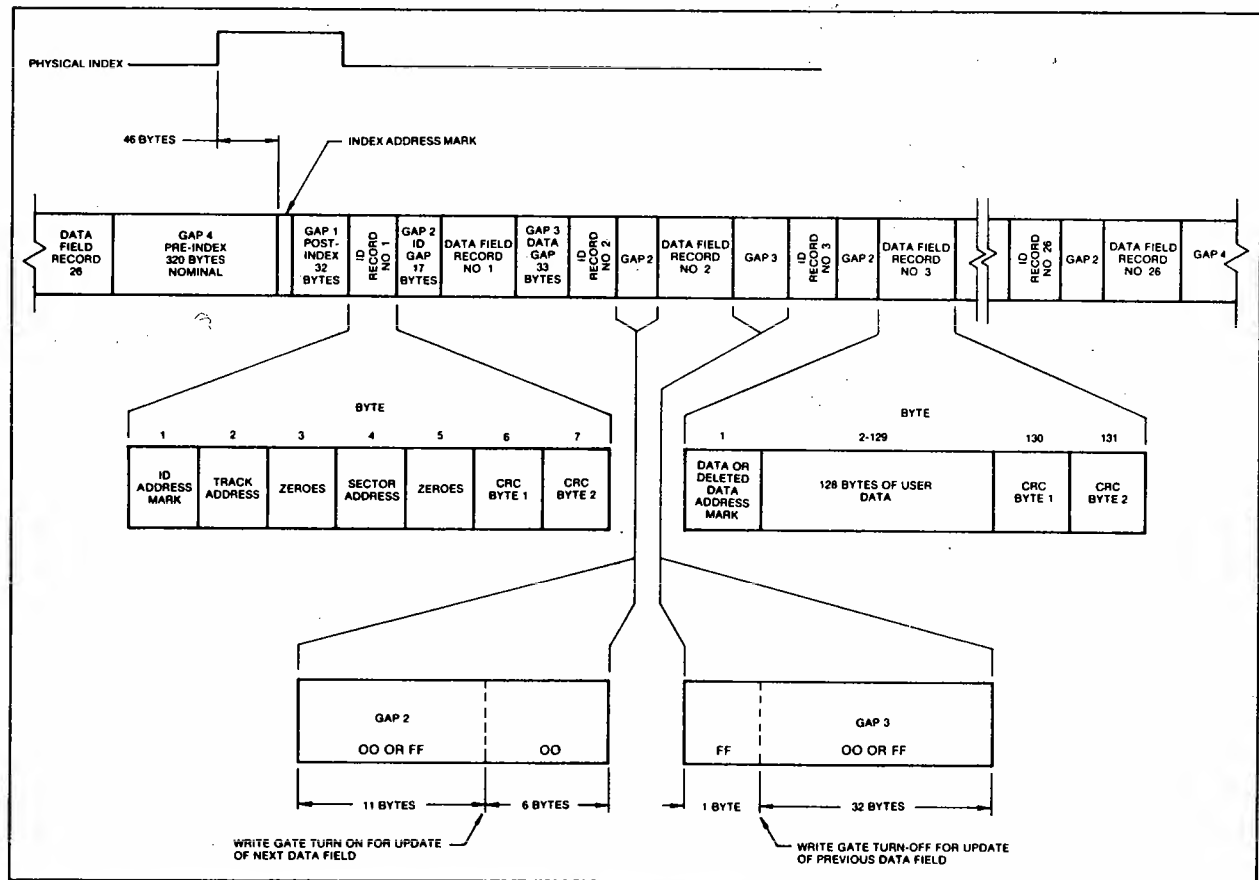
Non-IBM Formats

Non-IBM formats are very similar to the IBM formats except a different algorithm is used to ascertain the sector length from the sector length byte in the ID field. This permits a wide range of sector lengths from 16 to 4096 bytes. Refer to Section V, Type II commands with b flag equal to zero. Note that F7 through FE must not appear in the sector length byte of the ID field.

In formatting the FD1771, only two requirements regarding GAP sizes must be met. GAP 2 (i.e., the gap between the ID field and data field) must be 17 bytes of which the last 6 bytes must be zero and that every address mark be preceded by at least one byte of zeros. However, it is recommended that every GAP be at least 17 bytes long with 6 bytes of zeros. The FD1771 does not require the index address mark (i.e., DATA = FC, CLK = D7) and need not be present.

References:

- 1) IBM Diskette OEM Information GA21-9190-1.
- 2) SA900 IBM Compatibility Reference Manual — Shugart Associates.



TRACK FORMAT

ELECTRICAL CHARACTERISTICS**Maximum Ratings**

V_{DD} with respect to V_{BB} (Ground)	+20 to -0.3V
Max Voltage to any input with respect to V_{BB}	+20 to -0.3V
Operating Temperature	0°C to 70°C
Storage Temperature	-55°C to +125°C

OPERATING CHARACTERISTICS (DC)

T_A	0°C to 70°C, $V_{DD} = +12.0V \pm .6V$, $V_{BB} = -5.0 \pm .5V$, $V_{SS} = 0V$, $V_{CC} = +5V \pm .25V$ $I_{DD} = 10$ ma Nominal, $I_{CC} = 30$ ma Nominal, $P_{BB} = 0.4 \mu a$ Nominal
-------	--

Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
I_{LI}	Input Leakage			10	μA	$V_{IN} = V_{DD}$
I_{LO}	Output Leakage			10	μA	$V_{OUT} = V_{DD}$
V_{IH}	Input High Voltage	2.6			V	
V_{IL}	Input Low Voltage (All Inputs)			0.8	V	
V_{OH}	Output High Voltage	2.8			V	$I_O = -100 \mu A$
V_{OL}	Output Low Voltage			0.45	V	$I_O = 1.0$ mA

TIMING CHARACTERISTICS

T_A	0°C to 70°C, $V_{DD} = +12V \pm .6V$, $V_{BB} = -5V \pm .25V$, $V_{SS} = 0V$, $V_{CC} = +5V \pm .25V$
-------	---

NOTE: Timings are given for 2 MHz Clock. For those timings noted, values will double when chip is operated at 1 MHz. Use 1 MHz when using mini-floppy.

Read Operations

Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TSET	Setup ADDR and CS to \overline{RE}	100			nsec	$C_L = 25$ pf
THLD	Hold ADDR and CS from \overline{RE}	10			nsec	
TRE	\overline{RE} Pulse Width	450			nsec	
TDRR	DRQ Reset from \overline{RE}			750	nsec	
TIRR	INTRQ Reset from \overline{RE}			3000	nsec	$C_L = 25$ pf
TDACC	Data Access from \overline{RE}			450	nsec	
TDOH	Data Hold from \overline{RE}	50		150	nsec	$C_L = 25$ pf

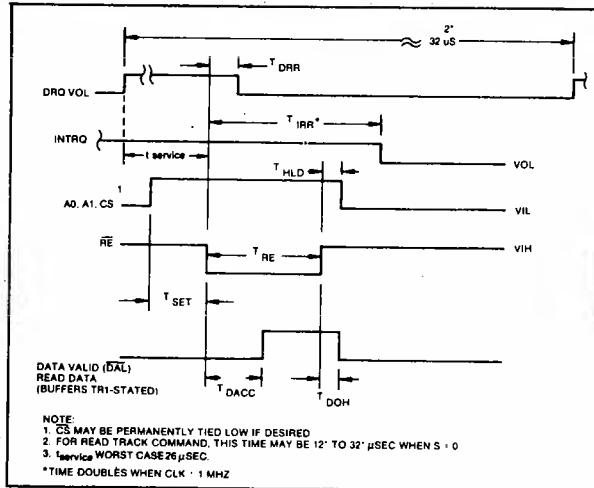
Write Operations

Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TSET	Setup ADDR and CS to \overline{WE}	100			nsec	See Note
THLD	Hold ADDR and CS from \overline{WE}	10			nsec	
TWE	\overline{WE} Pulse Width	450	300		nsec	
TDRR	DRQ Reset from \overline{WE}			750	nsec	
TIRR	INTRQ Reset from \overline{WE}			3000	nsec	
TDS	Data Setup to \overline{WE}	350			nsec	
TDH	Data Hold from \overline{WE}	150			nsec	

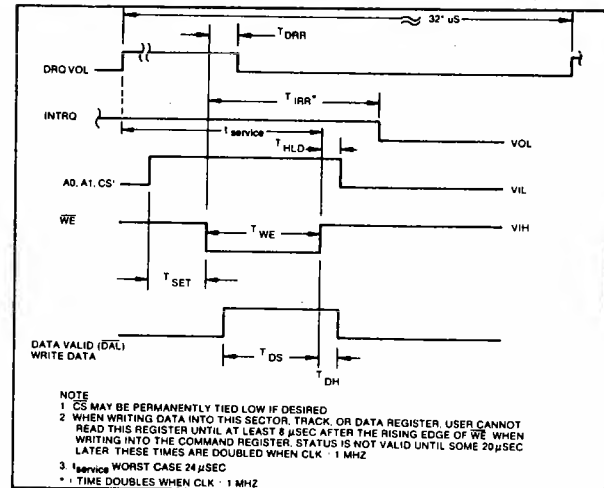
External Data Separation ($\overline{XTDS} = 0$)

Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TPWX	Pulse Width Read Data & Read Clock	150		350	nsec	
TCX	Clock Cycle External	2500			nsec	
TDEX	Data to Clock	500			nsec	
TDDX	Data to Data Cycle	2500			nsec	

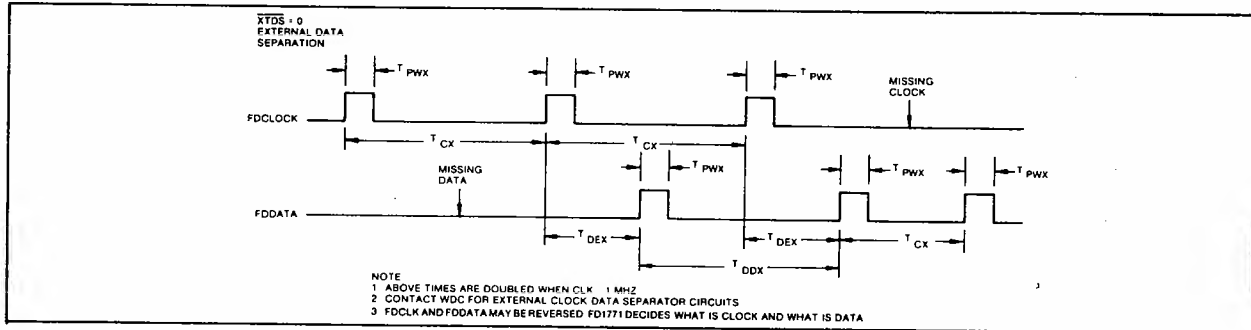
FD1771-01



READ ENABLE TIMING



WRITE ENABLE TIMING



READ TIMING (XTDS = 0)

Internal Data Separation (XTDS = 1)

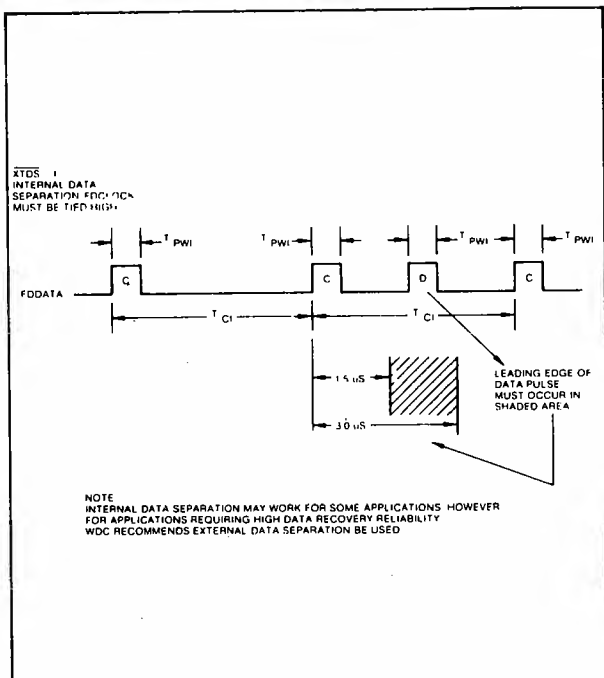
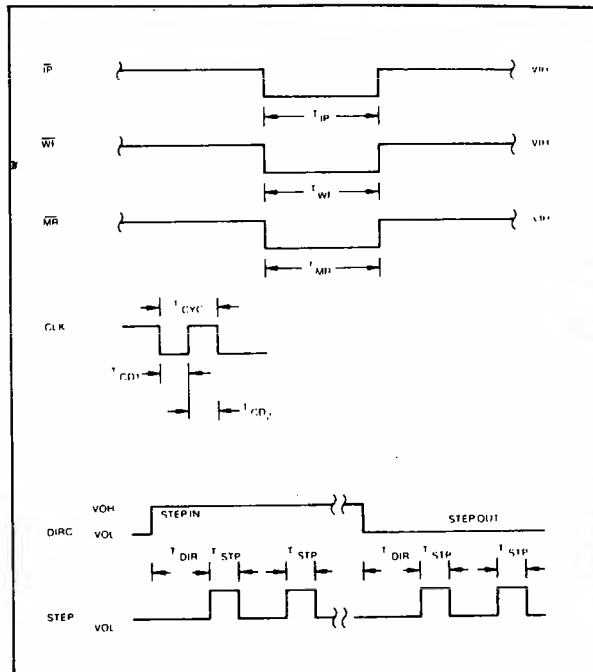
Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TPWI	Pulse Width Data and Clock	150		1000	nsec	
TCI	Clock Cycle Internal	3500		5000	nsec	

Write Data Timing

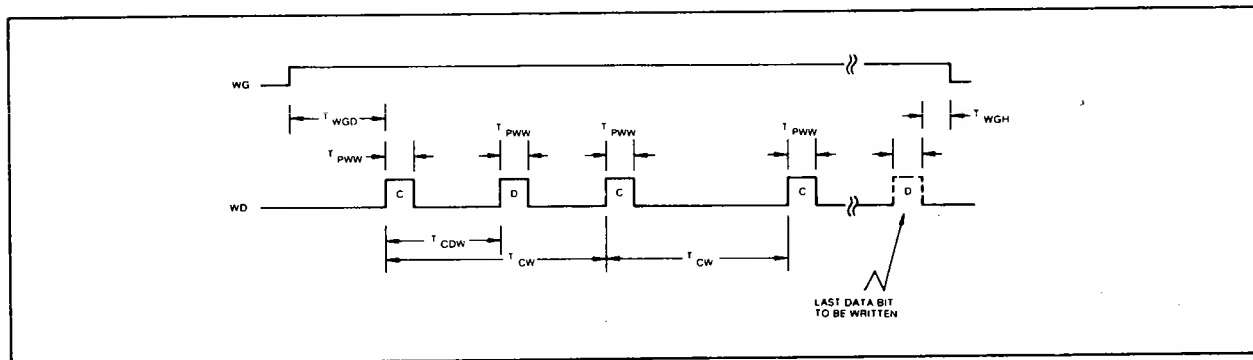
Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TWGD	Write Gate to Data		1200		nsec	300 nsec \pm CLK tolerance
TPWW	Pulse Width Write Data	500		600	nsec	
TCDW	Clock to Data		2000		nsec	
TCW	Clock Cycle Write		4000		nsec	
TWGH	Write Gate Hold to Data	0		100	nsec	\pm CLK tolerance

Miscellaneous Timing

Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TCD1	Clock Duty	175			nsec	2 MHz \pm 1% See Note These times doubled when CLK = 1 MHz
TCD2	Clock Duty	210			nsec	
TSTP	Step Pulse Output	3800		4200	nsec	
TDIR	Direct Setup to Step	24			nsec	
TMR	Master Reset Pulse Width	10			nsec	
TIP	Index Pulse Width	10			nsec	
TWF	Write Fault Pulse Width	10			nsec	

READ TIMING ($\overline{XTDS} = 1$)

MISCELLANEOUS TIMING



WRITE DATA TIMING

See page 725 for ordering information.

FD1771-01

Information furnished by Western Digital Corporation is believed to be accurate and reliable. However, no responsibility is assumed by Western Digital Corporation for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Western Digital Corporation. Western Digital Corporation reserves the right to change specifications at anytime without notice.

WESTERN DIGITAL

C O R P O R A T I O N

1771-01 Application Notes

1771-01

INTRODUCTION

The FD1771-01 Floppy Disk Formatter/Controller is a MOS/LSI device designed to ease the task of interfacing the 8" or 5¼ (mini-floppy) disk drive to a host processor. It is ideally suited for a wide range of microprocessors, providing an 8-bit bi-directional interface to the CPU for all control and data transfers. Requiring standard +12, ±5V power supplies, the 1771 is available in ceramic or plastic 40 pin dual-in-line packages.

The 1771 has been designed to be compatible with the IBM 3740 standard. This single-density Frequency Modulated (FM) recording technique, records a clock bit between a data bit serially on each track. Figure 1 illustrates how a HEX "D2" is recorded. Note that when the data bit to be written is zero, no pulse or flux transition is recorded. For the 8" drive, there are 77 tracks, with 26 sectors on each track. Each sector contains 128 bytes of data. Although there is no "standard" format for the mini-floppy, most manufacturers utilize either 35 or 40 tracks per side, with 16 sectors of 128 bytes each per track. Both the 8" and 5¼" formats must be soft-sectored, i.e., there are no physical holes to denote sector locations. The hard-sectored disk has been losing popularity, mainly due to the fact that the sector lengths cannot be increased.

Being soft-sector compatible, the 1771 must know where each sector begins on the track. This is performed by using Address Marks. These bytes are recorded on the disk with certain clock pulses missing, and are unique from all other data and gap bytes recorded on the track. Six distinct Address Marks can be used:

Description	Data	Clock Pattern
Index Address Mark	FC	D7
ID Address Mark	FE	C7
Data Address Mark	FB	C7
User defined	FA	C7
User Defined	F9	C7
Deleted Address Mark	F8	C7

The two "User Defined" Address Marks are unique to the 1771, and do not appear in the IBM 3740 standard. These Address Marks can be used to

define the type of data i.e., "object" or "text" data, alternate sector data, or any other purpose the user chooses.

PROCESSOR INTERFACE

The 1771 contains five internal registers that can be accessed via the 8-bit DAL lines by the CPU. These registers are used to control the movement of the head, read and write sectors, and perform all other functions at the drive. Regardless of the operation performed, it must be initiated through one or more of these registers. They are selected by a proper binary code on the A0, A1 lines in conjunction with the \overline{RE} and \overline{WE} lines when the device is selected. The registers and their addresses are:

\overline{CS}	A ₁	A ₀	$\overline{RE} = 0$	$\overline{WE} = 0$
0	0	0	STATUS REG	COMMAND REG
0	0	1	TRACK REG	TRACK REG
0	1	0	SECTOR REG	SECTOR REG
0	1	1	DATA REG	DATA REG
1	X	X	Deselected	Deselected

Command Register: This is a write-only register used to send all commands to the 1771.

Status Register: This is a read-only register that must be read at the completion of every command to determine whether execution was successful. It may also be used to monitor command execution, and to sense when data is required by the drive for read or write operations.

Track Register: This R/W register holds the current position of the R/W head.

Sector Register: This R/W register holds the desired sector number for read and write commands.

Data Register: This R/W register contains the data to be read or written to a particular sector.

INTERRUPTS

There are two INTERRUPT lines for CPU use. These are the DRQ (Data Request) and INTRQ (Interrupt Request). These are active high, open drain outputs and require a pull-up resistor of 10K or greater to +5V. Both of these signals also appear in the status register as the Busy (INTRQ) and the data request (DRQ) bits. The user has the option of utilizing these hardware lines for system interrupts, or through

software by polling the status register. The choice is dependent upon the particular microprocessor and support hardware of the system.

INTRQ: This line is used to signify the completion of any command. It is reset low when a new command is loaded into the command register, or when the status register is read.

DRQ: This line is active high whenever the data register requires servicing. During a read command, it signifies that the data register contains a byte of data from the disk and may be read by the CPU. During a write command, it signifies that the data register is empty and may be loaded with the next byte to be written on the disk. The DRQ line is reset whenever the data register is read or written to. It is also reset when a new command is loaded into the command register, providing the new command is not a Forced Interrupt, and the 1771 is not busy (Busy Bit = 0).

WRITE SECTOR

With the use of the WRITE SECTOR command, the CPU can access any desired sector(s) in a track. Prior to loading this command, the R/W head of the drive must be positioned over the specific track. This can be first accomplished with the use of any of the Type I commands. Once positioned, the CPU must load the desired sector number into the sector register, then issue the command. The head will load, and the 1771 will begin searching for the correct ID field. If the correct sector and track is not found within 2 revolutions of the disk, the RECORD-NOT-FOUND bit will be set in the status register, and the command will be terminated. Once found, the 1771 will issue a DRQ in request of the first data byte to be written. Once the data register is loaded, the 1771 will issue a DRQ for each byte to be recorded, until the entire sector is written. For the 8" drive, the user must load the data register 24 microseconds after a DRQ is generated. Failure to meet this time will cause the lost data bit to be set, and a byte of zeros substituted and written on the disk.

READ SECTOR

The READ SECTOR command functions in much the same way as the WRITE SECTOR command. The sector register must again be loaded with the desired sector number, before the read command can be loaded. After the ID field has been found, the 1771 will begin generating DRQ's, with the data register being loaded with each byte of the sector field. For the 8" drive, the user must read the data register at least 26 microseconds after the DRQ is generated. Failure to meet this time will cause the lost data bit to be set in the status register, while the next assembled byte will overwrite the contents of the data register.

Both the Read and Write sector commands also

contain an "m" flag for accessing multiple sectors. The sector register is incremented internally after each sector is read or written to. Eventually the sector register will exceed the physical number of sectors on the track. The user can either issue the Forced Interrupt command after the last sector, or wait for the 1771 to interrupt out. In the latter case, the RECORD-NOT-FOUND status bit will be set.

FLOPPY DISK INTERFACE

For the most part, the actual Floppy Disk Interface will consist mainly of Buffer/Drivers. Most drives manufactured today require an open collector TTL interface, with appropriate resistor terminal networks. Figure 2 shows the interface of the 1771 to a Shugart SA400 Drive. Aside from the data separator, the interface consists mainly of 7438's and 7414 TTL gates. A 9602 one-shot is used for the desired head load delay. In this illustration, the 6800 microprocessor is used via a 6820 Peripheral Interface Adapter to control all functions of the 1771. Similarly, other parallel port devices (such as the 8255 for 8080 systems) can be used for the interface, or the 1771 may simply be tied directly to the systems data bus and control lines, providing TTL loading factors are observed.

DATA SEPERATION

The internal DATA SEPERATOR of the 1771 can be used by tying the XTDS line high, and supplying the combined clock and data pulses on the FD data line. In order to maintain an error rate better than 1 in 10^8 , and external data separator is recommended.

Since the 1771 system clock is at 2 MHz, this allows for a 500 ns resolution. The internal data window will move 500 ns with respect to the incoming data bit. On the inner tracks of the drive, the bit shift is more severe and may occasionally cause a data or clock bit to fall outside of this data window. Since the 1771 will perform up to 5 retries, this error rate may be acceptable for some applications.

When the XTDS line is forced low, the 1771 will accept separated clock and data on the FDCLOCK and FDDATA lines. Figure 3 illustrates the timing of these signals. The actual FDCLOCK and FDDATA lines may be reversed; the 1771 will determine which line is clock and which is data when an Address Mark is detected. This feature greatly simplifies the design of the data separator.

Figure 4 illustrates the Phase-Lock Loop method for data separation. The circuit operates at 8 MHz, or 32 times the frequency of a received bit cell. The MC4024 VCO is used to supply the nominal clock frequency. The first 74LS161 counter provides a divide by 16 frequency and a carry to one side of the MC4044 phase detector. The other input of the MC4044 is tied to another 74LS161 counter which is affected by the incoming data stream. The output of

the phase detector is a signal proportional to the differences of the incoming pulses. This is then fed through a low pass filter, and to the input of the MC4024 to adjust the output frequency. Figures 5 thru 8 illustrate other types of data separators.

These employ the "Counter Separator" techniques and are quite different from the Phase-Lock-Loop method. With the addition of "One-Shot" delay element or an input clock, most of the complexity of the PPL circuit can be eliminated.

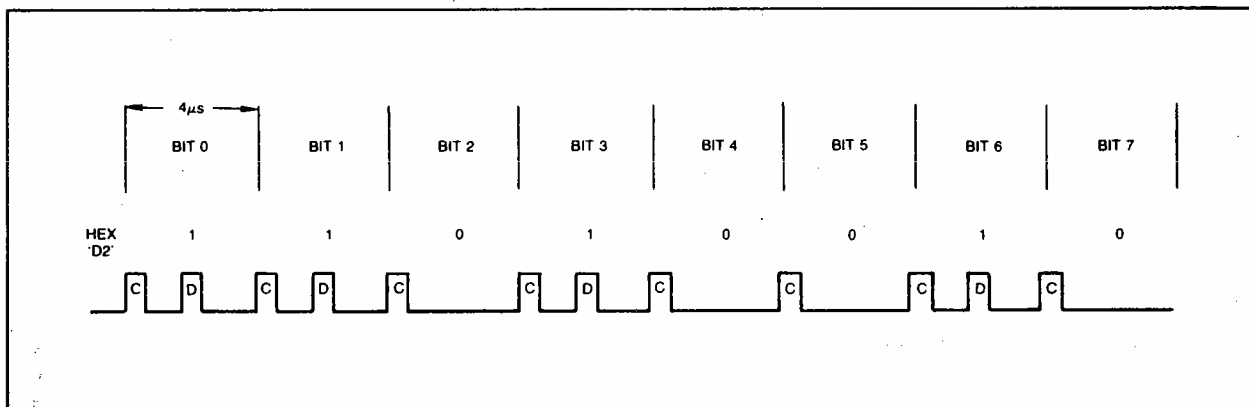


FIGURE 1. FM RECORDING.

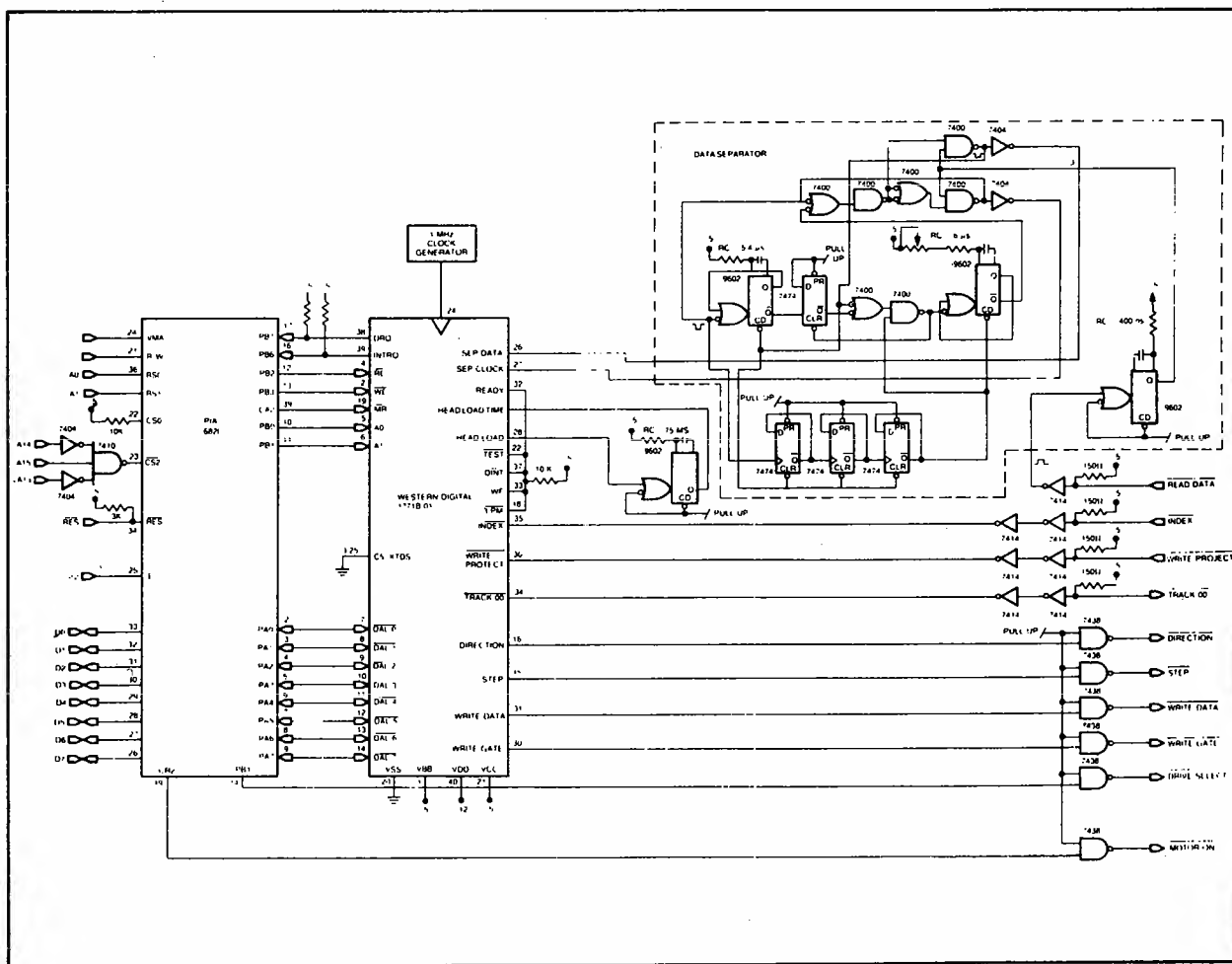


FIGURE 2. 1771 TO SHUGART SA400 DRIVE

1771-01

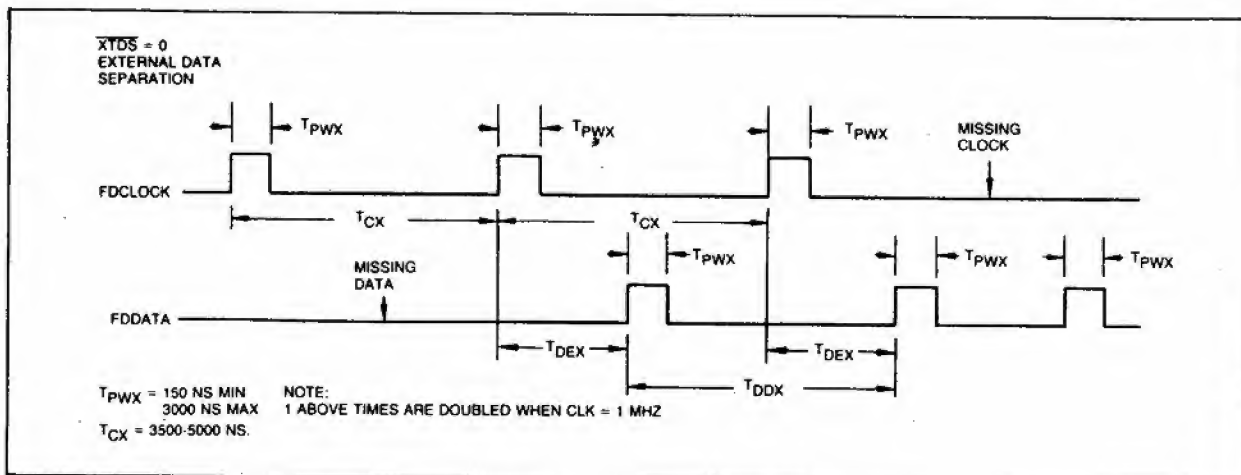


FIGURE 3. EXTERNAL DATA SEPARATOR TIMING.

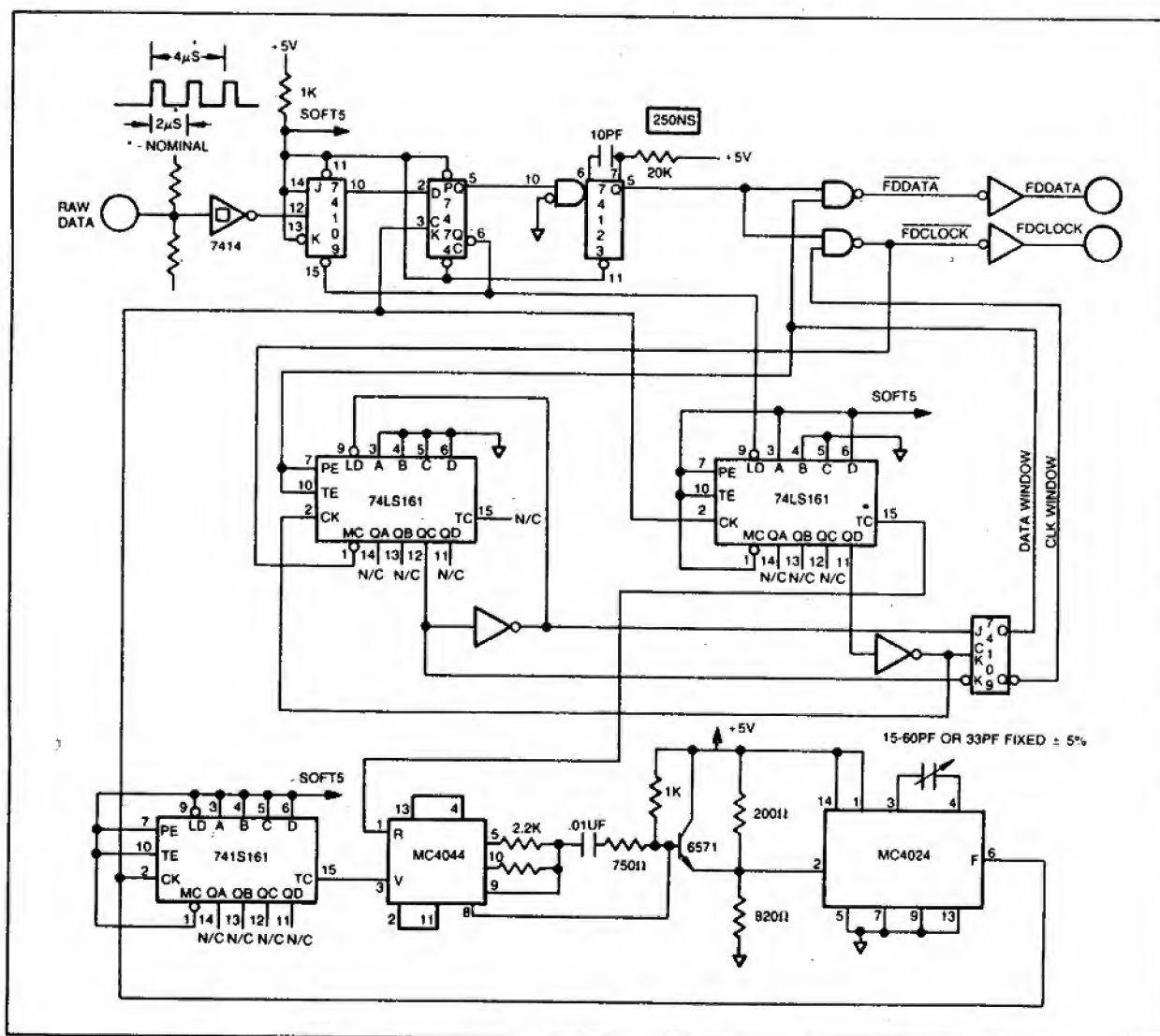


FIGURE 4. CIRCUIT PROVIDED COURTESY OF MOTOROLA AND ICOM CORPS.

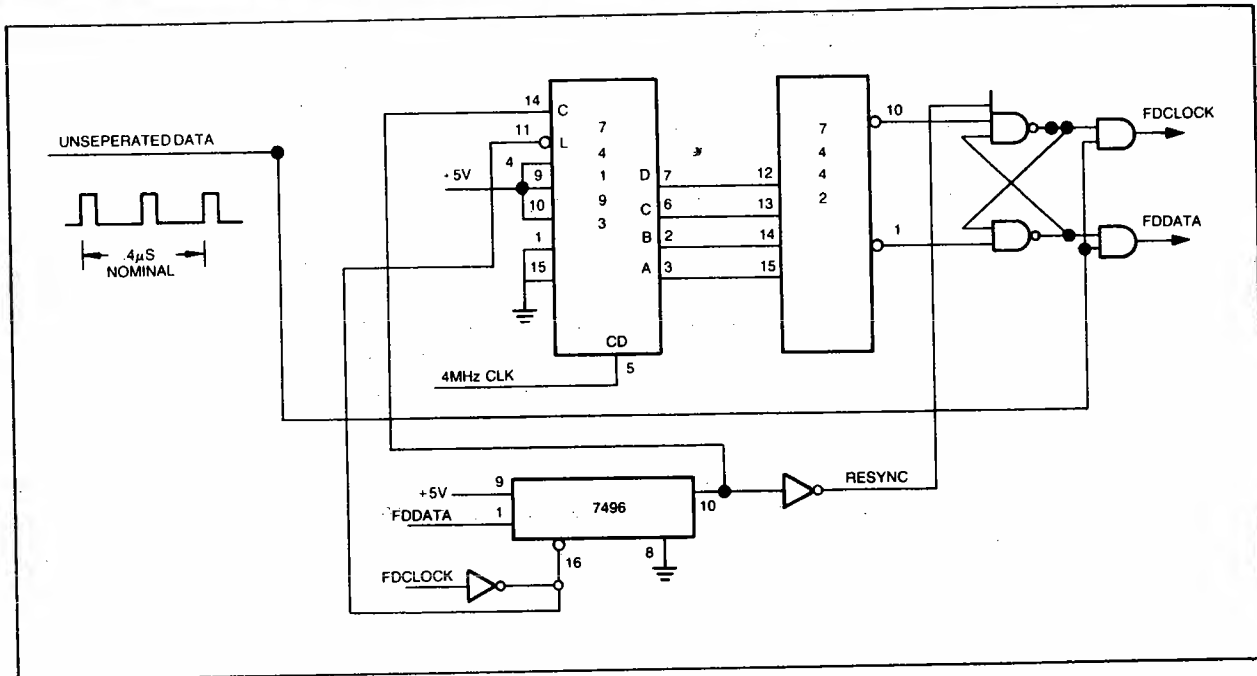


FIGURE 5. CIRCUIT PROVIDED COURTESY OF PROCESSOR APPLICATIONS LTD.

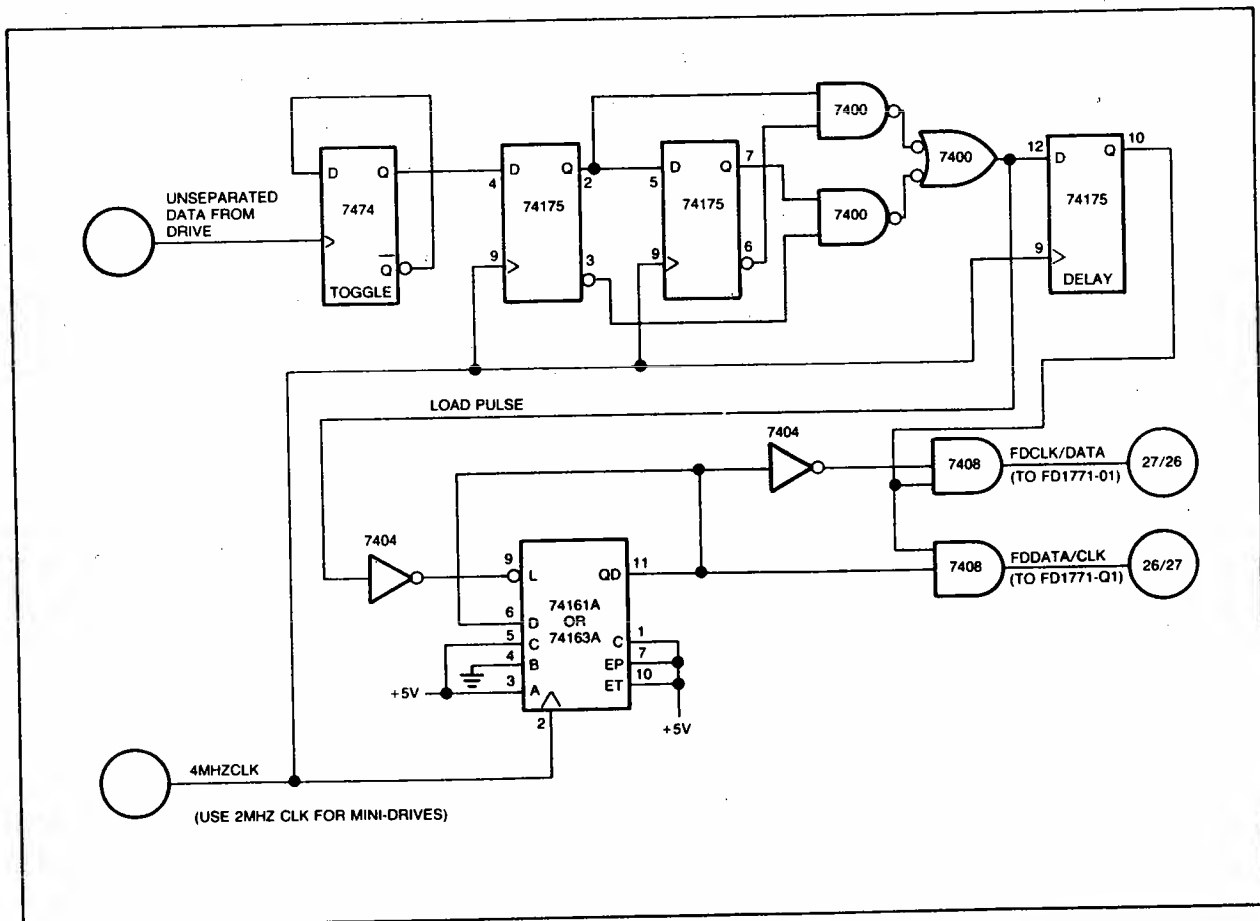


FIGURE 6.

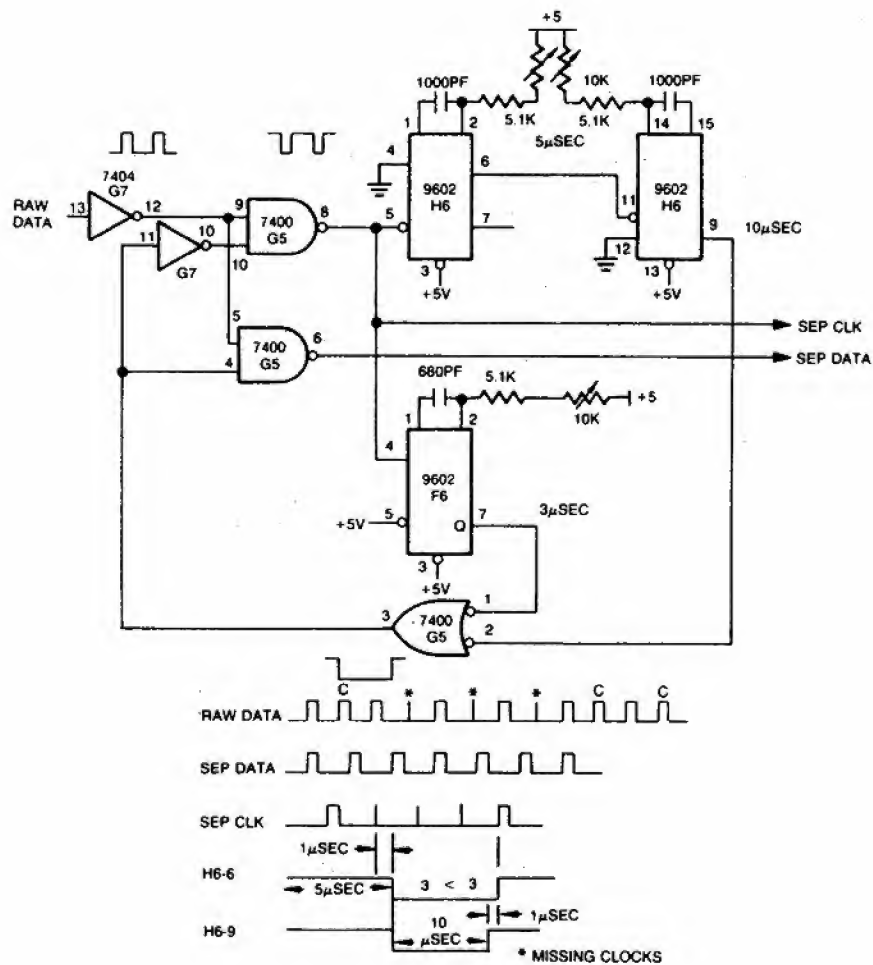


FIGURE 7. CIRCUIT PROVIDED COURTESY OF ACUTEST CORP.

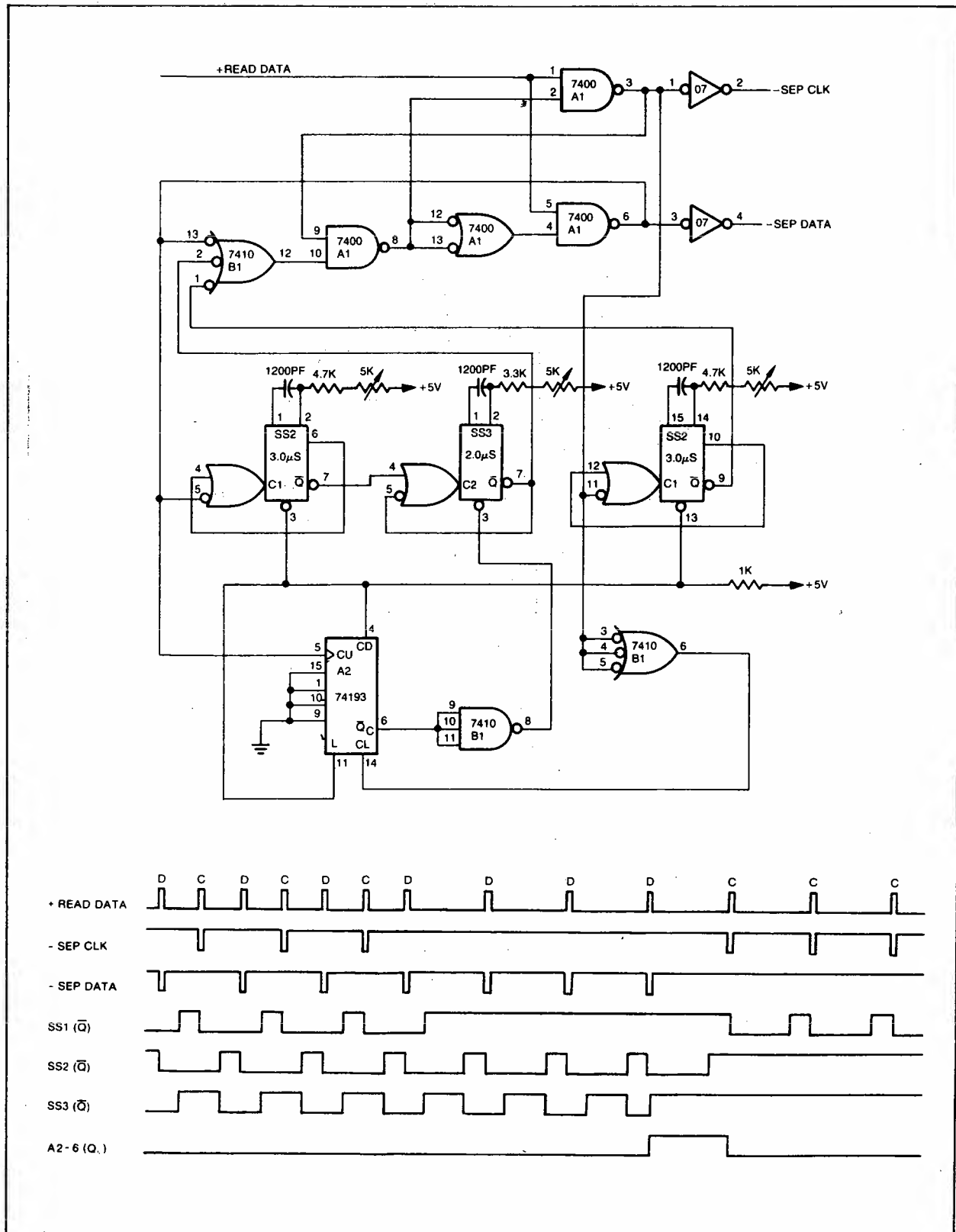


FIGURE 8. CIRCUIT PROVIDED COURTESY OF SHUGART ASSOCIATES.

1771-01

Information furnished by Western Digital Corporation is believed to be accurate and reliable. However, no responsibility is assumed by Western Digital Corporation for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Western Digital Corporation. Western Digital Corporation reserves the right to change specifications at anytime without notice.

WESTERN DIGITAL

C O R P O R A T I O N

FD179X-02

Floppy Disk Formatter/Controller Family

FD179X-02

FEATURES

- TWO VFO CONTROL SIGNALS — RG & VFOE
- SOFT SECTOR FORMAT COMPATIBILITY
- AUTOMATIC TRACK SEEK WITH VERIFICATION
- ACCOMMODATES SINGLE AND DOUBLE DENSITY FORMATS
 - IBM 3740 Single Density (FM)
 - IBM System 34 Double Density (MFM)
 - Non IBM Format for Increased Capacity
- READ MODE
 - Single/Multiple Sector Read with Automatic Search or Entire Track Read
 - Selectable 128, 256, 512 or 1024 Byte Sector Lengths
- WRITE MODE
 - Single/Multiple Sector Write with Automatic Sector Search
 - Entire Track Write for Diskette Formatting
- SYSTEM COMPATIBILITY
 - Double Buffering of Data 8 Bit Bi-Directional Bus for Data, Control and Status
 - DMA or Programmed Data Transfers
 - All Inputs and Outputs are TTL Compatible
 - On-Chip Track and Sector Registers/Comprehensive Status Information

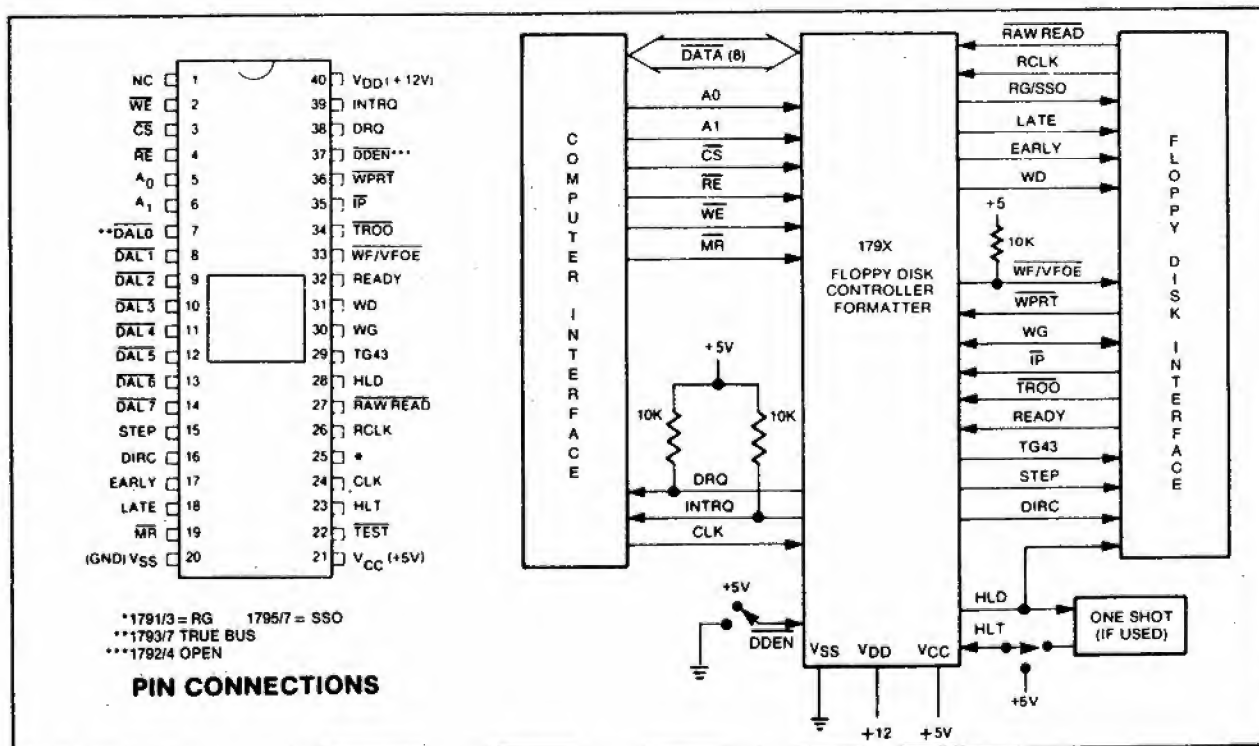
- PROGRAMMABLE CONTROLS
 - Selectable Track to Track Stepping Time
 - Side Select Compare
- INTERFACES TO WD1691 DATA SEPARATOR
- WINDOW EXTENSION
- INCORPORATES ENCODING/DECODING AND ADDRESS MARK CIRCUITRY
- FD1792/4 IS SINGLE DENSITY ONLY
- FD1795/7 HAS A SIDE SELECT OUTPUT

179X-02 FAMILY CHARACTERISTICS

FEATURES	1791	1792	1793	1794	1795	1797
Single Density (FM)	X	X	X	X	X	X
Double Density (MFM)	X		X		X	X
True Data Bus			X	X		X
Inverted Data Bus	X	X			X	
Write Precomp	X	X	X	X	X	X
Side Selection Output					X	X

APPLICATIONS

8" FLOPPY AND 5¼" MINI FLOPPY CONTROLLER
SINGLE OR DOUBLE DENSITY
CONTROLLER/FORMATTER



FD179X SYSTEM BLOCK DIAGRAM

FD179X-02

PIN OUTS

PIN NUMBER	PIN NAME	SYMBOL	FUNCTION																									
1	NO CONNECTION	NC	Pin 1 is internally connected to a back bias generator and must be left open by the user.																									
19	MASTER RESET	\overline{MR}	A logic low (50 microseconds min.) on this input resets the device and loads HEX 03 into the command register. The Not Ready (Status Bit 7) is reset during \overline{MR} ACTIVE. When \overline{MR} is brought to a logic high a RESTORE Command is executed, regardless of the state of the Ready signal from the drive. Also, HEX 01 is loaded into sector register.																									
20	POWER SUPPLIES	V_{ss}	Ground																									
21		V_{cc}	+5V \pm 5%																									
40		V_{dd}	+12V \pm 5%																									
COMPUTER INTERFACE:																												
2	WRITE ENABLE	\overline{WE}	A logic low on this input gates data on the DAL into the selected register when \overline{CS} is low.																									
3	CHIP SELECT	\overline{CS}	A logic low on this input selects the chip and enables computer communication with the device.																									
4	READ ENABLE	\overline{RE}	A logic low on this input controls the placement of data from a selected register on the DAL when \overline{CS} is low.																									
5,6	REGISTER SELECT LINES	A0, A1	These inputs select the register to receive/transfer data on the DAL lines under \overline{RE} and \overline{WE} control: <table><tr><td>\overline{CS}</td><td>A1</td><td>A0</td><td>\overline{RE}</td><td>\overline{WE}</td></tr><tr><td>0</td><td>0</td><td>0</td><td>Status Reg</td><td>Command Reg</td></tr><tr><td>0</td><td>0</td><td>1</td><td>Track Reg</td><td>Track Reg</td></tr><tr><td>0</td><td>1</td><td>0</td><td>Sector Reg</td><td>Sector Reg</td></tr><tr><td>0</td><td>1</td><td>1</td><td>Data Reg</td><td>Data Reg</td></tr></table>	\overline{CS}	A1	A0	\overline{RE}	\overline{WE}	0	0	0	Status Reg	Command Reg	0	0	1	Track Reg	Track Reg	0	1	0	Sector Reg	Sector Reg	0	1	1	Data Reg	Data Reg
\overline{CS}	A1	A0	\overline{RE}	\overline{WE}																								
0	0	0	Status Reg	Command Reg																								
0	0	1	Track Reg	Track Reg																								
0	1	0	Sector Reg	Sector Reg																								
0	1	1	Data Reg	Data Reg																								
7-14	DATA ACCESS LINES	DAL0-DAL7	Eight bit Bidirectional bus used for transfer of data, control, and status. This bus is receiver enabled by \overline{WE} or transmitter enabled by \overline{RE} . Each line will drive 1 standard TTL load.																									
24	CLOCK	CLK	This input requires a free-running 50% duty cycle square wave clock for internal timing reference, 2 MHz \pm 1% for 8" drives, 1 MHz \pm 1% for mini-floppies.																									
38	DATA REQUEST	DRQ	This open drain output indicates that the DR contains assembled data in Read operations, or the DR is empty in Write operations. This signal is reset when serviced by the computer through reading or loading the DR in Read or Write operations, respectively. Use 10K pull-up resistor to +5.																									
39	INTERRUPT REQUEST	INTRQ	This open drain output is set at the completion of any command and is reset when the STATUS register is read or the command register is written to. Use 10K pull-up resistor to +5.																									
FLOPPY DISK INTERFACE:																												
15	STEP	STEP	The step output contains a pulse for each step.																									
16	DIRECTION	DIRC	Direction Output is active high when stepping in, active low when stepping out.																									
17	EARLY	EARLY	Indicates that the WRITE DATA pulse occurring while Early is active (high) should be shifted early for write precompensation.																									
18	LATE	LATE	Indicates that the write data pulse occurring while Late is active (high) should be shifted late for write precompensation.																									

PIN NUMBER	PIN NAME	SYMBOL	FUNCTION
22	TEST	TEST	This input is used for testing purposes only and should be tied to +5V or left open by the user unless interfacing to voice coil actuated steppers.
23	HEAD LOAD TIMING	HLT	When a logic high is found on the HLT input the head is assumed to be engaged. It is typically derived from a 1 shot triggered by HLD.
25	READ GATE (1791, 1792, 1793, 1794)	RG	This output is used for synchronization of external data separators. The output goes high after two Bytes of zeros in single density, or 4 Bytes of either zeros or ones in double density operation.
25	SIDE SELECT OUTPUT (1795, 1797)	SSO	The logic level of the Side Select Output is directly controlled by the 'S' flag in Type II or III commands. When U = 1, SSO is set to a logic 1. When U = 0, SSO is set to a logic 0. The SSO is compared with the side information in the Sector I.D. Field. If they do not compare Status Bit 4 (RNF) is set. The Side Select Output is only updated at the beginning of a Type II or III command. It is forced to a logic 0 upon a MASTER RESET condition.
26	READ CLOCK	RCLK	A nominal square-wave clock signal derived from the data stream must be provided to this input. Phasing (i.e. RCLK transitions) relative to RAW READ is important but polarity (RCLK high or low) is not.
27	RAW READ	RAW READ	The data input signal directly from the drive. This input shall be a negative pulse for each recorded flux transition.
28	HEAD LOAD	HLD	The HLD output controls the loading of the Read-Write head against the media.
29	TRACK GREATER THAN 43	TG43	This output informs the drive that the Read/Write head is positioned between tracks 44-76. This output is valid only during Read and Write Commands.
30	WRITE GATE	WG	This output is made valid before writing is to be performed on the diskette.
31	WRITE DATA	WD	A 200 ns (MFM) or 500 ns (FM) output pulse per flux transition. WD contains the unique Address marks as well as data and clock in both FM and MFM formats.
32	READY	READY	This input indicates disk readiness and is sampled for a logic high before Read or Write commands are performed. If Ready is low the Read or Write operation is not performed and an interrupt is generated. Type I operations are performed regardless of the state of Ready. The Ready input appears in inverted format as Status Register bit 7.
33	WRITE FAULT VFO ENABLE	WF/VFOE	This is a bi-directional signal used to signify writing faults at the drive, and to enable the external PLO data separator. When WG = 1, Pin 33 functions as a WF input. If WF = 0, any write command will immediately be terminated. When WG = 0, Pin 33 functions as a VFOE output. VFOE will go low during a read operation after the head has loaded and settled (HLT = 1). On the 1795/7, it will remain low until the last bit of the second CRC byte in the ID field. VFOE will then go high until 8 bytes (MFM) or 4 bytes (FM) before the Address Mark. It will then go active until the last bit of the second CRC byte of the Data Field. On the 1791/3, VFOE will remain low until the end of the Data Field. This pin has an internal 100K Ohm pull-up resistor.
34	TRACK 00	TR00	This Input informs the FD179X that the Read/Write head is positioned over Track 00.

PIN NUMBER	PIN NAME	SYMBOL	FUNCTION
35	INDEX PULSE	IP	This input informs the FD179X when the index hole is encountered on the diskette.
36	WRITE PROTECT	WPRT	This input is sampled whenever a Write Command is received. A logic low terminates the command and sets the Write Protect Status bit.
37	DOUBLE DENSITY	DDEN	This input pin selects either single or double density operation. When DDEN = 0, double density is selected. When DDEN = 1, single density is selected. This line must be left open on the 1792/4.

GENERAL DESCRIPTION

The FD179X are N-Channel Silicon Gate MOS LSI devices which perform the functions of a Floppy Disk Formatter/Controller in a single chip implementation. The FD179X, which can be considered the end result of both the FD1771 and FD1781 designs, is IBM 3740 compatible in single density mode (FM) and System 34 compatible in Double Density Mode (MFM). The FD179X contains all the features of its predecessor the FD1771, plus the added features necessary to read/write and format a double density diskette. These include address mark detection, FM and MFM encode and decode logic, window extension, and write precompensation. In order to maintain compatibility, the FD1771, FD1781, and FD179X designs were made as close as possible with the computer interface, instruction set, and I/O registers being identical. Also, head load control is identical. In each case, the actual pin assignments vary by only a few pins from any one to another.

The processor interface consists of an 8-bit bi-directional bus for data, status, and control word transfers. The FD179X is set up to operate on a multiplexed bus with other bus-oriented devices.

The FD179X is TTL compatible on all inputs and outputs. The outputs will drive ONE TTL load or three LS loads. The 1793 is identical to the 1791 except the DAL lines are TRUE for systems that utilize true data busses.

The 1795/7 has a side select output for controlling double sided drives, and the 1792 and 1794 are "Single Density Only" versions of the 1791 and 1793 respectively. On these devices, DDEN must be left open.

ORGANIZATION

The Floppy Disk Formatter block diagram is illustrated on page 5. The primary sections include the parallel processor interface and the Floppy Disk Interface.

Data Shift Register — This 8-bit register assembles serial data from the Read Data Input (RAW READ) during Read operations and transfers serial data to the Write Data output during Write operations.

Data Register — This 8-bit register is used as a holding register during Disk Read and Write operations. In Disk Read operations the assembled data byte is transferred in parallel to the Data Register from the Data Shift Register. In Disk Write operations information is transferred in parallel from the Data Register to the Data Shift Register.

When executing the Seek command the Data Register holds the address of the desired Track position. This register is loaded from the DAL and gated onto the DAL under processor control.

Track Register — This 8-bit register holds the track number of the current Read/Write head position. It is incremented by one every time the head is stepped in (towards track 76) and decremented by one when the head is stepped out (towards track 00). The contents of the register are compared with the recorded track number in the ID field during disk Read, Write, and Verify operations. The Track Register can be loaded from or transferred to the DAL. This Register should not be loaded when the device is busy.

Sector Register (SR) — This 8-bit register holds the address of the desired sector position. The contents of the register are compared with the recorded sector number in the ID field during disk Read or Write operations. The Sector Register contents can be loaded from or transferred to the DAL. This register should not be loaded when the device is busy.

Command Register (CR) — This 8-bit register holds the command presently being executed. This register should not be loaded when the device is busy unless the new command is a force interrupt. The command register can be loaded from the DAL, but not read onto the DAL.

Status Register (STR) — This 8-bit register holds device Status information. The meaning of the Status bits is a function of the type of command previously executed. This register can be read onto the DAL, but not loaded from the DAL.

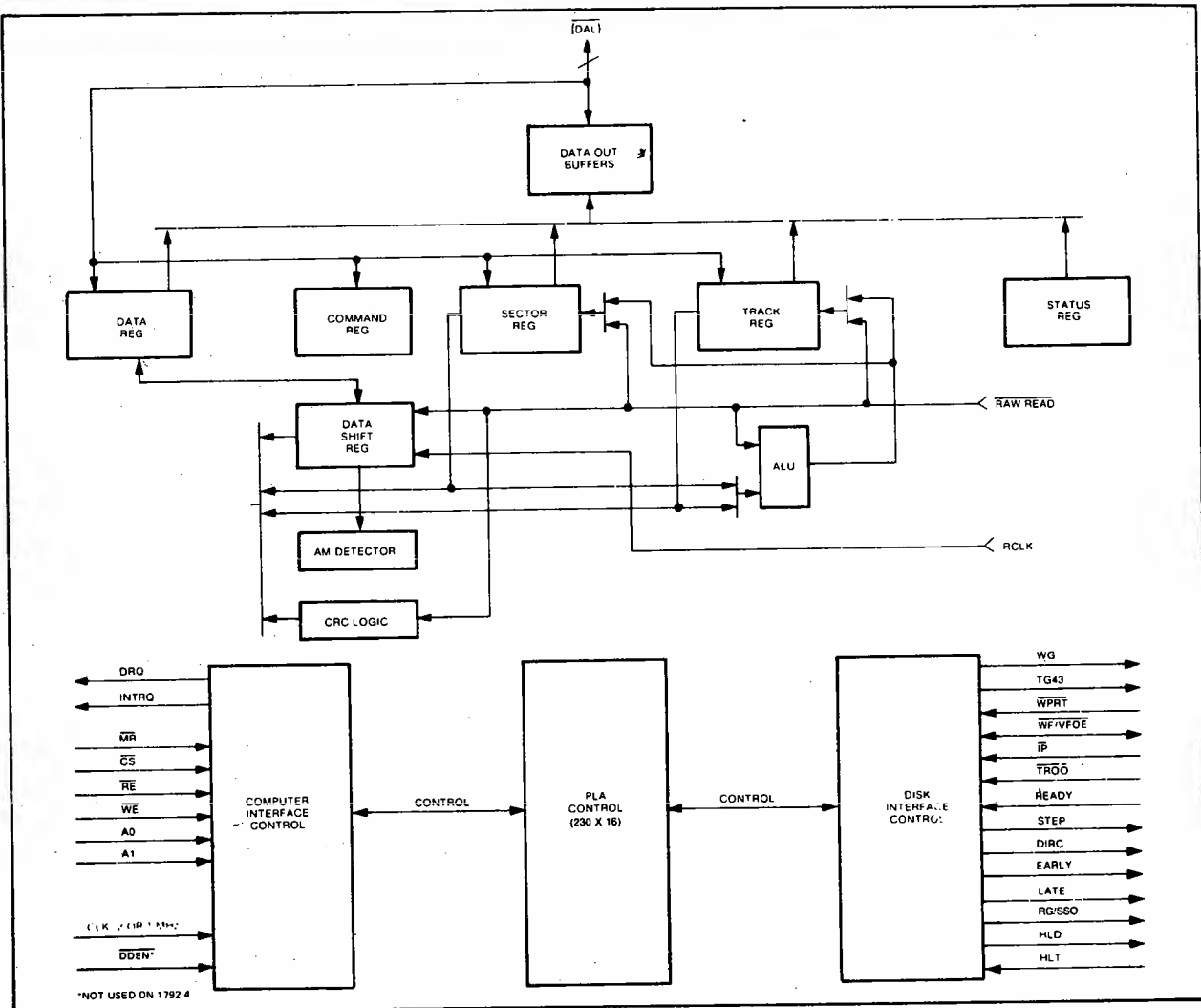
CRC Logic — This logic is used to check or to generate the 16-bit Cyclic Redundancy Check (CRC). The polynomial is: $G(x) = x^{16} + x^{12} + x^5 + 1$.

The CRC includes all information starting with the address mark and up to the CRC characters. The CRC register is preset to ones prior to data being shifted through the circuit.

Arithmetic/Logic Unit (ALU) — The ALU is a serial comparator, incrementer, and decrements and is used for register modification and comparisons with the disk recorded ID field.

Timing and Control — All computer and Floppy Disk Interface controls are generated through this logic. The internal device timing is generated from an external crystal clock.

The FD179X has two different modes of operation according to the state of DDEN. When DDEN = 0 double density (MFM) is assumed. When DDEN = 1, single



FD179X BLOCK DIAGRAM

density (FM) is assumed. 1792 & 1794 are single density only.

AM Detector — The address mark detector detects ID, data and index address marks during read and write operations.

PROCESSOR INTERFACE

The interface to the processor is accomplished through the eight Data Access Lines (DAL) and associated control signals. The DAL are used to transfer Data, Status, and Control words out of, or into the FD179X. The DAL are three state buffers that are enabled as output drivers when Chip Select (CS) and Read Enable (RE) are active (low logic state) or act as input receivers when CS and Write Enable (WE) are active.

When transfer of data with the Floppy Disk Controller is required by the host processor, the device address is decoded and CS is made low. The address bits A1 and A0, combined with the signals RE during a Read operation or WE during a Write operation are interpreted as selecting the following registers:

A1 - A0	READ (RE)	WRITE (WE)
0 0	Status Register	Command Register
0 1	Track Register	Track Register
1 0	Sector Register	Sector Register
1 1	Data Register	Data Register

During Direct Memory Access (DMA) types of data transfers between the Data Register of the FD179X and the processor, the Data Request (DRQ) output is used in Data Transfer control. This signal also appears as status bit 1 during Read and Write operations.

On Disk Read operations the Data Request is activated (set high) when an assembled serial input byte is transferred in parallel to the Data Register. This bit is cleared when the Data Register is read by the processor. If the Data Register is read after one or more characters are lost, by having new data transferred into the register prior to processor readout, the Lost Data bit is set in the Status Register. The Read operation continues until the end of sector is reached.

On Disk Write operations the data Request is activated when the Data Register transfers its contents to the Data

Shift Register, and requires a new data byte. It is reset when the Data Register is loaded with new data by the processor. If new data is not loaded at the time the next serial byte is required by the Floppy Disk, a byte of zeroes is written on the diskette and the Lost Data bit is set in the Status Register.

At the completion of every command an INTRQ is generated. INTRQ is reset by either reading the status register or by loading the command register with a new command. In addition, INTRQ is generated if a Force Interrupt command condition is met.

The 179X has two modes of operation according to the state of DDEN (Pin 37). When DDEN = 1, single density is selected. In either case, the CLK input (Pin 24) is at 2 MHz. However, when interfacing with the mini-floppy, the CLK input is set at 1 MHz for both single density and double density.

GENERAL DISK READ OPERATIONS

Sector lengths of 128, 256, 512 or 1024 are obtainable in either FM or MFM formats. For FM, DDEN should be placed to logical "1." For MFM formats, DDEN should be placed to a logical "0." Sector lengths are determined at format time by the fourth byte in the "ID" field.

Sector Length Table*	
Sector Length Field (hex)	Number of Bytes in Sector (decimal)
00	128
01	256
02	512
03	1024

*1795/97 may vary — see command summary.

The number of sectors per track as far as the FD179X is concerned can be from 1 to 255 sectors. The number of tracks as far as the FD179X is concerned is from 0 to 255 tracks. For IBM 3740 compatibility, sector lengths are 128 bytes with 26 sectors per track. For System 34 compatibility (MFM), sector lengths are 256 bytes/sector with 26 sectors/track; or lengths of 1024 bytes/sector with 8 sectors/track. (See Sector Length Table)

For read operations in 8" double density the FD179X requires RAW READ Data (Pin 27) signal which is a 200 ns pulse per flux transition and a Read clock (RCLK) signal to indicate flux transition spacings. The RCLK (Pin 26) signal is provided by some drives but if not it may be derived externally by Phase lock loops, one shots, or counter techniques. In addition, a Read Gate Signal is provided as an output (Pin 25) on 1791/92/93/94 which can be used to inform phase lock loops when to acquire synchronization. When reading from the media in FM, RG is made true when 2 bytes of zeroes are detected. The FD179X must find an address mark within the next 10 bytes; otherwise RG is reset and the search for 2 bytes of zeroes begins all over again. If an address mark is found within 10 bytes, RG remains true as long as the FD179X is deriving any useful information from the data stream. Similarly for MFM, RG is made active when 4 bytes of "00" or "FF" are detected. The FD179X must find an address mark within the next 16 bytes, otherwise RG is reset and search resumes.

During read operations (WG = 0), the VFOE (Pin 33) is provided for phase lock loop synchronization. VFOE will go active low when:

- Both HLT and HLD are True
- Settling Time, if programmed, has expired
- The 179X is inspecting data off the disk

If WF/VFOE is not used, leave open or tie to a 10K resistor to +5.

GENERAL DISK WRITE OPERATION

When writing is to take place on the diskette the Write Gate (WG) output is activated, allowing current to flow into the Read/Write head. As a precaution to erroneous writing the first data byte must be loaded into the Data Register in response to a Data Request from the FD179X before the Write Gate signal can be activated.

Writing is inhibited when the Write Protect input is a logic low, in which case any Write command is immediately terminated, an interrupt is generated and the Write Protect status bit is set. The Write Fault Input, when activated, signifies a writing fault condition detected in disk drive electronics such as failure to detect write current flow when the Write Gate is activated. On detection of this fault the FD179X terminates the current command, and sets the Write Fault bit (bit 5) in the Status Word. The Write Fault Input should be made inactive when the Write Gate output becomes inactive.

For write operations, the FD179X provides Write Gate (Pin 30) and Write Data (Pin 31) outputs. Write data consists of a series of 500 ns pulses in FM (DDEN = 1) and 200 ns pulses in MFM (DDEN = 0). Write Data provides the unique address marks in both formats.

Also during write, two additional signals are provided for write precompensation. These are EARLY (Pin 17) and LATE (Pin 18). EARLY is active true when the WD pulse appearing on (Pin 30) is to be written EARLY. LATE is active true when the WD pulse is to be written LATE. If both EARLY and LATE are low when the WD pulse is present, the WD pulse is to be written at nominal. Since write precompensation values vary from disk manufacturer to disk manufacturer, the actual value is determined by several one shots or delay lines which are located external to the FD179X. The write precompensation signals EARLY and LATE are valid for the duration of WD in both FM and MFM formats.

READY

Whenever a Read or Write command (Type II or III) is received the FD179X samples the Ready input. If this input is logic low the command is not executed and an interrupt is generated. All Type I commands are performed regardless of the state of the Ready input. Also, whenever a Type II or III command is received, the TG43 signal output is updated.

COMMAND DESCRIPTION

The FD179X will accept eleven commands. Command words should only be loaded in the Command Register when the Busy status bit is off (Status bit 0). The one exception is the Force Interrupt command. Whenever a command is being executed, the Busy status bit is set. When a command is completed, an interrupt is generated and the Busy status bit is reset. The Status Register indicates whether the completed command encountered an error or was fault free. For ease of discussion, commands are divided into four types. Commands and types are summarized in Table 1.

TABLE 1. COMMAND SUMMARY

A. Commands for Models: 1791, 1792, 1793, 1794

B. Commands for Models: 1795, 1797

Type	Command	Bits								Bits							
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
I	Restore	0	0	0	0	h	V	r1	r0	0	0	0	0	h	V	r1	r0
I	Seek	0	0	0	1	h	V	r1	r0	0	0	0	1	h	V	r1	r0
I	Step	0	0	1	T	h	V	r1	r0	0	0	1	T	h	V	r1	r0
I	Step-in	0	1	0	T	h	V	r1	r0	0	1	0	T	h	V	r1	r0
I	Step-out	0	1	1	T	h	V	r1	r0	0	1	1	T	h	V	r1	r0
II	Read Sector	1	0	0	m	S	E	C	0	1	0	0	m	L	E	U	0
II	Write Sector	1	0	1	m	S	E	C	a0	1	0	1	m	L	E	U	a0
III	Read Address	1	1	0	0	0	E	0	0	1	1	0	0	0	E	U	0
III	Read Track	1	1	1	0	0	E	0	0	1	1	1	0	0	E	U	0
III	Write Track	1	1	1	1	0	E	0	0	1	1	1	1	0	E	U	0
IV	Force Interrupt	1	1	0	1	l3	l2	l1	l0	1	1	0	1	l3	l2	l1	l0

FLAG SUMMARY

TABLE 2. FLAG SUMMARY

Command Type	Bit No(s)		Description																				
I	0, 1	r1 r0 = Stepping Motor Rate See Table 3 for Rate Summary																					
I	2	V = Track Number Verify Flag	V = 0, No verify V = 1, Verify on destination track																				
I	3	h = Head Load Flag	h = 1, Load head at beginning h = 0, Unload head at beginning																				
I	4	T = Track Update Flag	T = 0, No update T = 1, Update track register																				
II	0	a0 = Data Address Mark	a0 = 0, FB (DAM) a0 = 1, F8 (deleted DAM)																				
II	1	C = Side Compare Flag	C = 0, Disable side compare C = 1, Enable side compare																				
II & III	1	U = Update SSO	U = 0, Update SSO to 0 U = 1, Update SSO to 1																				
II & III	2	E = 15 MS Delay	E = 0, No 15 MS delay E = 1, 15 MS delay																				
II	3	S = Side Compare Flag	S = 0, Compare for side 0 S = 1, Compare for side 1																				
II	3	L = Sector Length Flag	<table><tr><td></td><td colspan="4">LSB's Sector Length in ID Field</td></tr><tr><td></td><td>00</td><td>01</td><td>10</td><td>11</td></tr><tr><td>L = 0</td><td>256</td><td>512</td><td>1024</td><td>128</td></tr><tr><td>L = 1</td><td>128</td><td>256</td><td>512</td><td>1024</td></tr></table>		LSB's Sector Length in ID Field					00	01	10	11	L = 0	256	512	1024	128	L = 1	128	256	512	1024
	LSB's Sector Length in ID Field																						
	00	01	10	11																			
L = 0	256	512	1024	128																			
L = 1	128	256	512	1024																			
II	4	m = Multiple Record Flag	m = 0, Single record m = 1, Multiple records																				
IV	0-3	l _x = Interrupt Condition Flags l ₀ = 1 Not Ready To Ready Transition l ₁ = 1 Ready To Not Ready Transition l ₂ = 1 Index Pulse l ₃ = 1 Immediate Interrupt, Requires A Reset l ₃ -l ₀ = 0 Terminate With No Interrupt (INTRQ)																					

*NOTE: See Type IV Command Description for further information.

TYPE I COMMANDS

The Type I Commands include the Restore, Seek, Step, Step-In, and Step-Out commands. Each of the Type I Commands contains a rate field (R1 R0), which determines the stepping motor rate as defined in Table 3.

A 2 μ s (MFM) or 4 μ s (FM) pulse is provided as an output to the drive. For every step pulse issued, the drive moves one track location in a direction determined by the direction output. The chip will step the drive in the same direction it last stepped unless the command changes the direction.

The Direction signal is active high when stepping in and low when stepping out. The Direction signal is valid 12 μ s before the first stepping pulse is generated.

The rates (shown in Table 3) can be applied to a Step-Direction Motor through the device interface.

TABLE 3. STEPPING RATES

CLK	2 MHz	2 MHz	1 MHz	1 MHz	2 MHz	1 MHz
DDEN	0	1	0	1	x	x
R1 R0	TEST=1	TEST=1	TEST=1	TEST=1	TEST=0	TEST=0
0 0	3 ms	3 ms	6 ms	6 ms	184 μ s	368 μ s
0 1	6 ms	6 ms	12 ms	12 ms	190 μ s	380 μ s
1 0	10 ms	10 ms	20 ms	20 ms	196 μ s	396 μ s
1 1	15 ms	15 ms	30 ms	30 ms	208 μ s	416 μ s

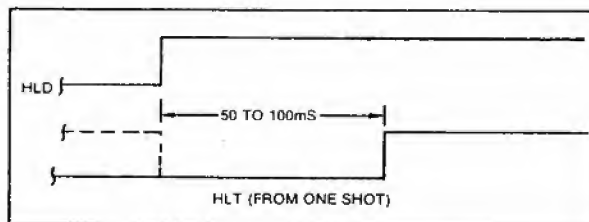
After the last directional step an additional 15 milliseconds of head settling time takes place if the Verify flag is set in Type I commands. Note that this time doubles to 30 ms for a 1 MHz clock. If $\overline{\text{TEST}} = 0$, there is zero settling time. There is also a 15 ms head settling time if the E flag is set in any Type II or III command.

When a Seek, Step or Restore command is executed an optional verification of Read-Write head position can be performed by settling bit 2 ($V = 1$) in the command word to a logic 1. The verification operation begins at the end of the 15 millisecond settling time after the head is loaded against the media. The track number from the first encountered ID Field is compared against the contents of the Track Register. If the track numbers compare and the ID Field Cyclic Redundancy Check (CRC) is correct, the verify operation is complete and an INTRQ is generated with no errors. If there is a match but not a valid CRC, the CRC error status bit is set (Status bit 3), and the next encountered ID field is read from the disk for the verification operation.

The FD179X must find an ID field with correct track number and correct CRC within 5 revolutions of the media; otherwise the seek error is set and an INTRQ is generated. If $V = 0$, no verification is performed.

The Head Load (HLD) output controls the movement of the read/write head against the media. HLD is activated at the beginning of a Type I command if the h flag is set ($h = 1$), at the end of the Type I command if the verify flag ($V = 1$), or upon receipt of any Type II or III command. Once HLD is active it remains active until either a Type I command is received with ($h = 0$ and $V = 0$); or if the FD179X is in an idle state (non-busy) and 15 index pulses have occurred.

Head Load timing (HLT) is an input to the FD179X which is used for the head engage time. When $\text{HLT} = 1$, the FD179X assumes the head is completely engaged. The head engage time is typically 30 to 100 ms depending on drive. The low to high transition on HLD is typically used to fire a one shot. The output of the one shot is then used for HLT and supplied as an input to the FD179X.



HEAD LOAD TIMING

When both HLD and HLT are true, the FD179X will then read from or write to the media. The "and" of HLD and HLT appears as status Bit 5 in Type I status.

In summary for the Type I commands: if $h = 0$ and $V = 0$, HLD is reset. If $h = 1$ and $V = 0$, HLD is set at the beginning of the command and HLT is not sampled nor is there an internal 15 ms delay. If $h = 0$ and $V = 1$, HLD is set near the end of the command, an internal 15 ms occurs, and the FD179X waits for HLT to be true. If $h = 1$ and $V = 1$, HLD is set at the beginning of the command. Near the end of the command, after all the steps have been issued, an internal 15 ms delay occurs and the FD179X then waits for HLT to occur.

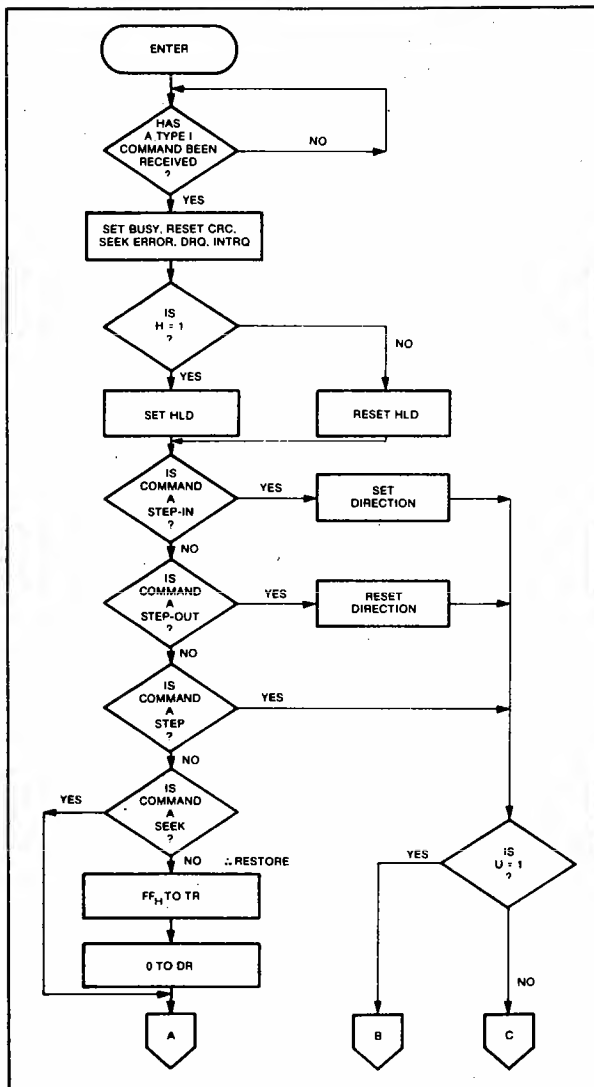
For Type II and III commands with E flag off, HLD is made active and HLT is sampled until true. With E flag on, HLD is made active, an internal 15 ms delay occurs and then HLT is sampled until true.

RESTORE (SEEK TRACK 0)

Upon receipt of this command the Track 00 ($\overline{\text{TR00}}$) input is sampled. If $\overline{\text{TR00}}$ is active low indicating the Read-Write head is positioned over track 0, the Track Register is loaded with zeroes and an interrupt is generated. If $\overline{\text{TR00}}$ is not active low, stepping pulses (pins 15 to 16) at a rate specified by the R1 R0 field are issued until the $\overline{\text{TR00}}$ input is activated. At this time the Track Register is loaded with zeroes and an interrupt is generated. If the $\overline{\text{TR00}}$ input does not go active low after 255 stepping pulses, the FD179X terminates operation, interrupts, and sets the Seek error status bit, providing the V flag is set. A verification operation also takes place if the V flag is set. The h bit allows the head to be loaded at the start of command. Note that the Restore command is executed when $\overline{\text{MR}}$ goes from an active to an inactive state and that the DRQ pin stays low.

SEEK

This command assumes that the Track Register contains the track number of the current position of the Read-Write head and the Data Register contains the desired track number. The FD179X will update the Track register and issue stepping pulses in the appropriate direction until the contents of the Track register are equal to the contents of



TYPE I COMMAND FLOW

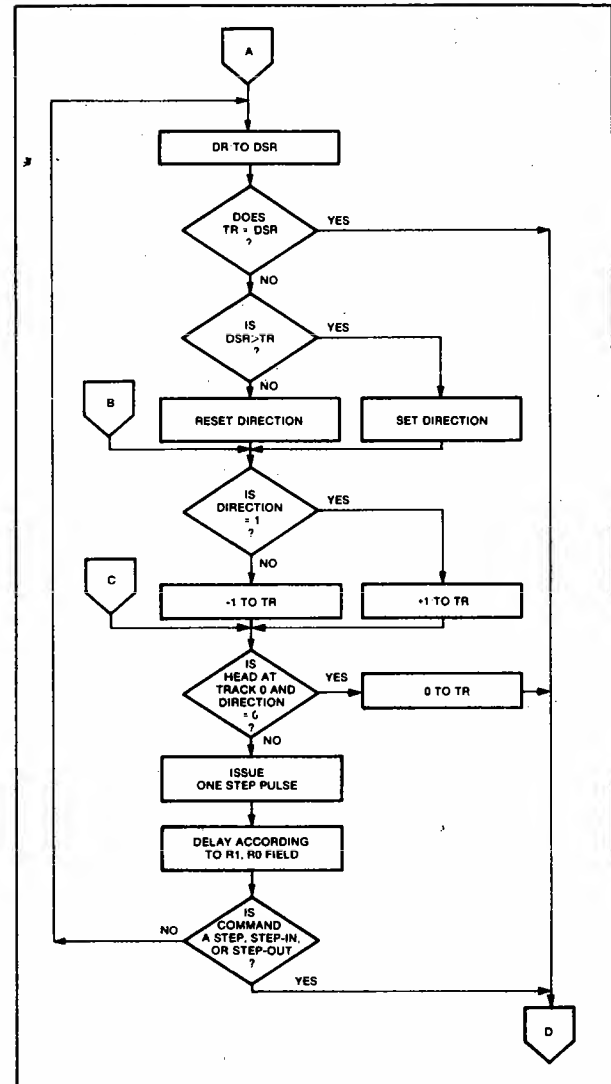
the Data Register (the desired track location). A verification operation takes place if the V flag is on. The h bit allows the head to be loaded at the start of the command. An interrupt is generated at the completion of the command. Note: When using multiple drives, the track register must be updated for the drive selected before seeks are issued.

STEP

Upon receipt of this command, the FD179X issues one stepping pulse to the disk drive. The stepping motor direction is the same as in the previous step command. After a delay determined by the $r1f0$ field, a verification takes place if the V flag is on. If the U flag is on, the Track Register is updated. The h bit allows the head to be loaded at the start of the command. An interrupt is generated at the completion of the command.

STEP-IN

Upon receipt of this command, the FD179X issues one stepping pulse in the direction towards track 76. If the U



TYPE I COMMAND FLOW

flag is on, the Track Register is incremented by one. After a delay determined by the $r1f0$ field, a verification takes place if the V flag is on. The h bit allows the head to be loaded at the start of the command. An interrupt is generated at the completion of the command.

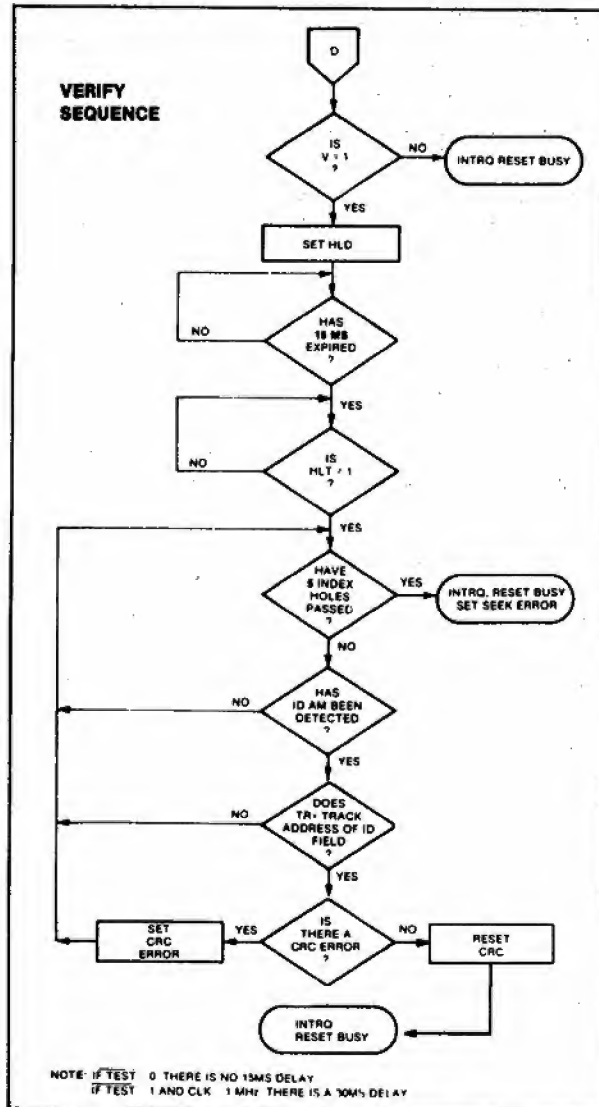
STEP-OUT

Upon receipt of this command, the FD179X issues one stepping pulse in the direction towards track 0. If the U flag is on, the Track Register is decremented by one. After a delay determined by the $r1f0$ field, a verification takes place if the V flag is on. The h bit allows the head to be loaded at the start of the command. An interrupt is generated at the completion of the command.

EXCEPTIONS

On the 1795/7 devices, the SSO output is not affected during Type 1 commands, and an internal side compare does not take place when the (V) Verify Flag is on.

FD179X-02



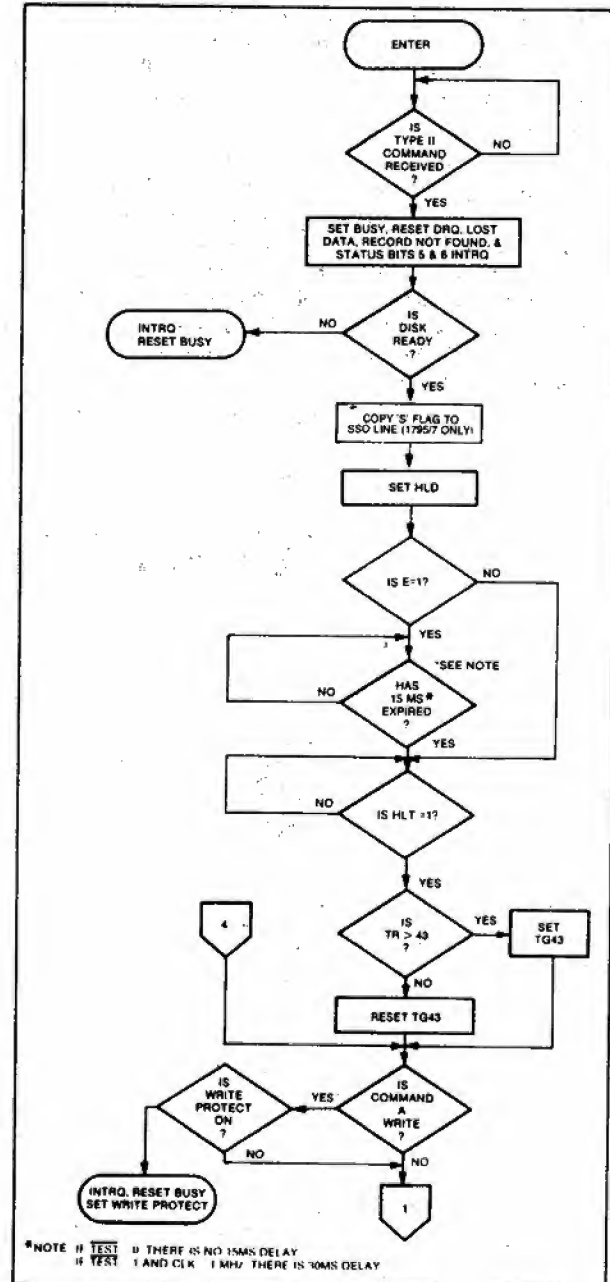
TYPE I COMMAND FLOW

TYPE II COMMANDS

The Type II Commands are the Read Sector and Write Sector commands. Prior to loading the Type II Command into the Command Register, the computer must load the Sector Register with the desired sector number. Upon receipt of the Type II command, the busy status Bit is set. If the E flag = 1 (this is the normal case) HLD is made active and HLT is sampled after a 15 msec delay. If the E flag is 0, the head is loaded and HLT sampled with no 15 msec delay. The ID field and Data Field format are shown on page 13.

When an ID field is located on the disk, the FD179X compares the Track Number on the ID field with the Track Register. If there is not a match, the next encountered ID field is read and a comparison is again made. If there was a match, the Sector Number of the ID field is compared with the Sector Register. If there is not a Sector match, the next encountered ID field is read off the disk and comparisons again made. If the ID field CRC is correct, the data field is

then located and will be either written into, or read from depending upon the command. The FD179X must find an ID field with a Track number, Sector number, side number, and CRC within four revolutions of the disk; otherwise, the Record not found status bit is set (Status bit 3) and the command is terminated with an interrupt.



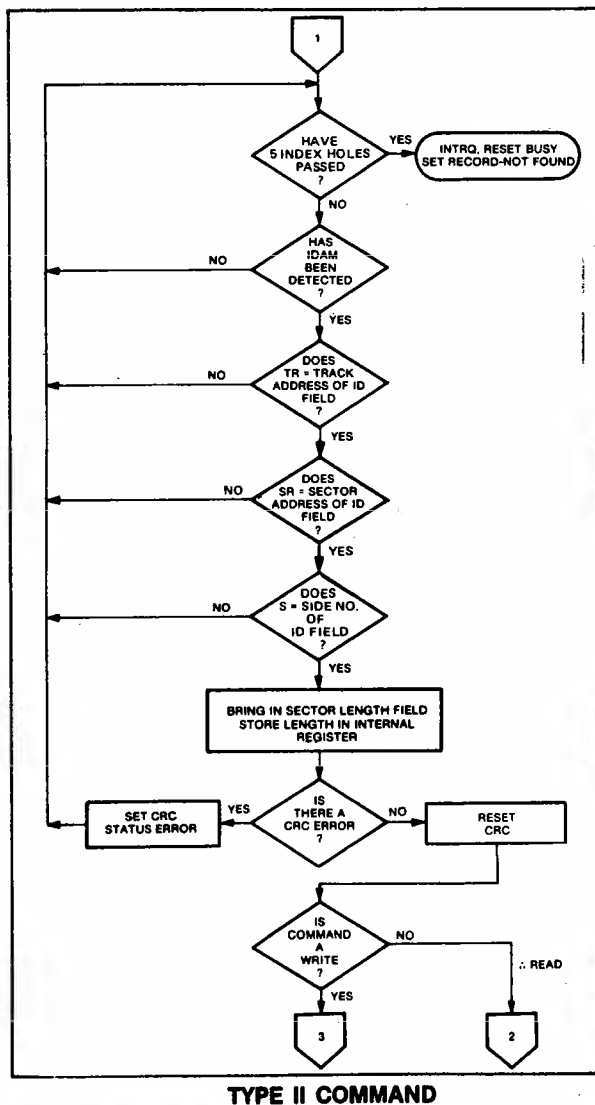
TYPE II COMMAND

Each of the Type II Commands contains an (m) flag which determines if multiple records (sectors) are to be read or written, depending upon the command. If $m = 0$, a single sector is read or written and an interrupt is generated at the completion of the command. If $m = 1$, multiple records are read or written with the sector register internally updated so that an address verification can occur on the next

record. The FD179X will continue to read or write multiple records and update the sector register in numerical ascending sequence until the sector register exceeds the number of sectors on the track or until the Force Interrupt command is loaded into the Command Register, which terminates the command and generates an interrupt.

For example: If the FD179X is instructed to read sector 27 and there are only 26 on the track, the sector register exceeds the number available. The FD179X will search for 5 disk revolutions, interrupt out, reset busy, and set the record not found status bit.

The Type II commands for 1791-94 also contain side select compare flags. When $C = 0$ (Bit 1) no side comparison is made. When $C = 1$, the LSB of the side number is read off the ID Field of the disk and compared with the contents of the (S) flag (Bit 3). If the S flag compares with the side number recorded in the ID field, the FD179X continues with the ID search. If a comparison is not made within 5 index pulses, the interrupt line is made active and the Record-Not-Found status bit is set.

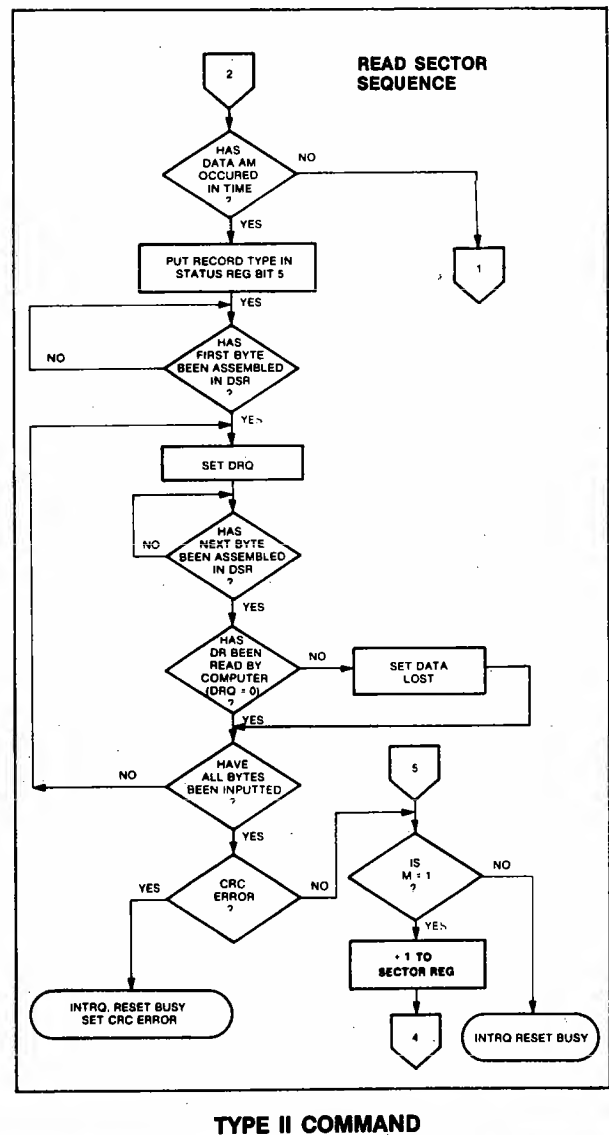


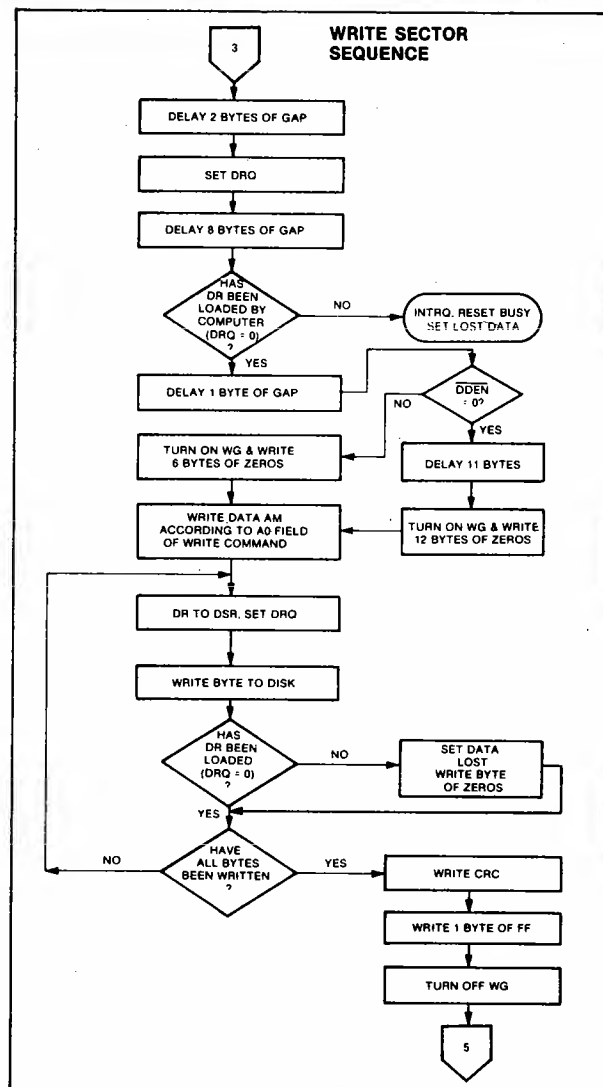
The Type II and III commands for the 1795-97 contain a side select flag (Bit 1). When $U = 0$, SSO is updated to 0. Similarly, $U = 1$ updates SSO to 1. The chip compares the SSO to the ID field. If they do not compare within 5 revolutions the interrupt line is made active and the RNF status bit is set.

The 1795/7 READ SECTOR and WRITE SECTOR commands include a 'L' flag. The 'L' flag, in conjunction with the sector length byte of the ID Field, allows different byte lengths to be implemented in each sector. For IBM compatibility, the 'L' flag should be set to a one.

READ SECTOR

Upon receipt of the Read Sector command, the head is loaded, the Busy status bit set, and when an ID field is encountered that has the correct track number, correct sector number, correct side number, and correct CRC, the data field is presented to the computer. The Data Address



**TYPE II COMMAND**

Mark of the data field must be found within 30 bytes in single density and 43 bytes in double density of the last ID field CRC byte; if not, the ID field is searched for and verified again followed by the Data Address Mark search. If after 5 revolutions the DAM cannot be found, the Record Not Found status bit is set and the operation is terminated. When the first character or byte of the data field has been shifted through the DSR, it is transferred to the DR, and DRQ is generated. When the next byte is accumulated in the DSR, it is transferred to the DR and another DRQ is generated. If the Computer has not read the previous contents of the DR before a new character is transferred that character is lost and the Lost Data Status bit is set. This sequence continues until the complete data field has been inputted to the computer. If there is a CRC error at the end of the data field, the CRC error status bit is set, and the command is terminated (even if it is a multiple record command).

At the end of the Read operation, the type of Data Address Mark encountered in the data field is recorded in the Status Register (Bit 5) as shown:

**STATUS
BIT 5**

1	Deleted Data Mark
0	Data Mark

WRITE SECTOR

Upon receipt of the Write Sector command, the head is loaded (HLD active) and the Busy status bit is set. When an ID field is encountered that has the correct track number, correct sector number, correct side number, and correct CRC, a DRQ is generated. The FD179X counts off 11 bytes in single density and 22 bytes in double density from the CRC field and the Write Gate (WG) output is made active if the DRQ is serviced (i.e., the DR has been loaded by the computer). If DRQ has not been serviced, the command is terminated and the Lost Data status bit is set. If the DRQ has been serviced, the WG is made active and six bytes of zeroes in single density and 12 bytes in double density are then written on the disk. At this time the Data Address Mark is then written on the disk as determined by the A0 field of the command as shown below:

A0 Data Address Mark (Bit 0)

1	Deleted Data Mark
0	Data Mark

The FD179X then writes the data field and generates DRQ's to the computer. If the DRQ is not serviced in time for continuous writing the Lost Data Status Bit is set and a byte of zeroes is written on the disk. The command is not terminated. After the last data byte has been written on the disk, the two-byte CRC is computed internally and written on the disk followed by one byte of logic ones in FM or in MFM. The WG output is then deactivated. For a 2 MHz clock the INTRQ will set 8 to 12 μ sec after the last CRC byte is written. For partial sector writing, the proper method is to write the data and fill the balance with zeroes. By letting the chip fill the zeroes, errors may be masked by the lost data status and improper CRC Bytes.

TYPE III COMMANDS**READ ADDRESS**

Upon receipt of the Read Address command, the head is loaded and the Busy Status Bit is set. The next encountered ID field is then read in from the disk, and the six data bytes of the ID field are assembled and transferred to the DR, and a DRQ is generated for each byte. The six bytes of the ID field are shown below:

TRACK ADDR	SIDE NUMBER	SECTOR ADDRESS	SECTOR LENGTH	CRC 1	CRC 2
1	2	3	4	5	6

Although the CRC characters are transferred to the computer, the FD179X checks for validity and the CRC error status bit is set if there is a CRC error. The Track Address of the ID field is written into the sector register so that a comparison can be made by the user. At the end of the operation an interrupt is generated and the Busy Status is reset.

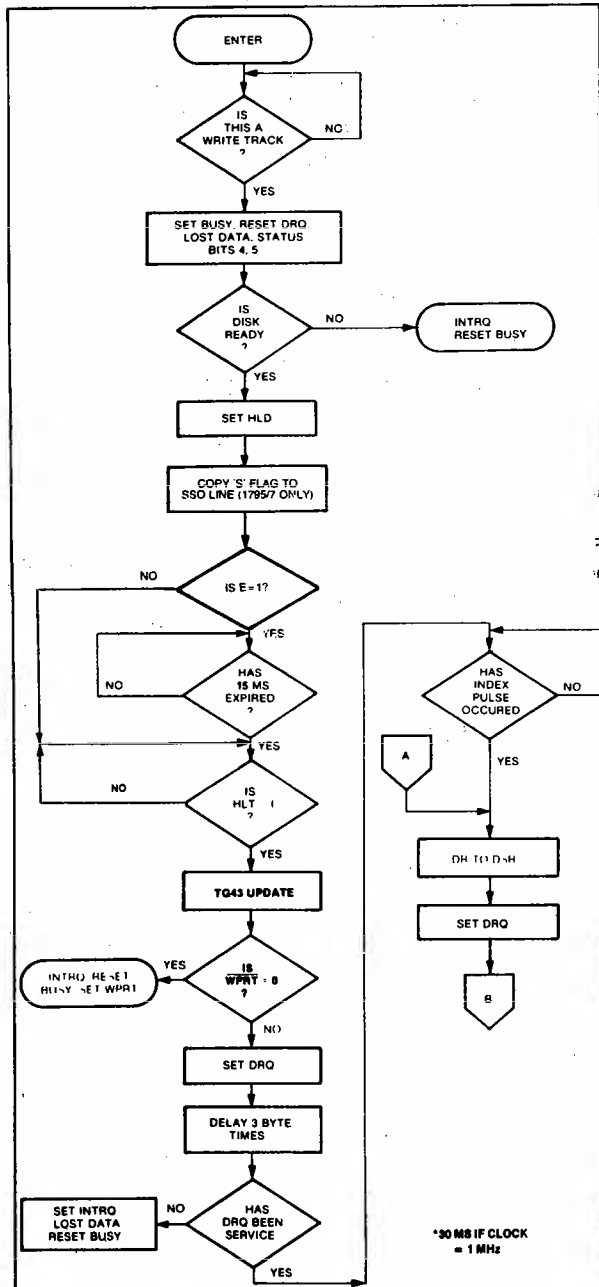
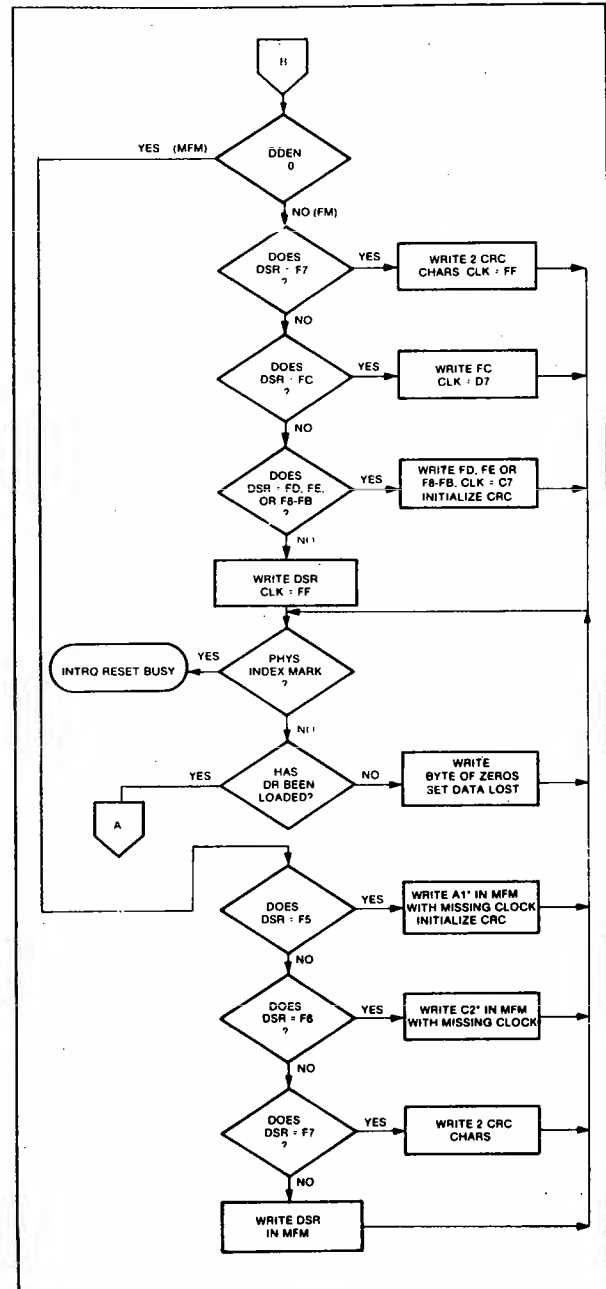
READ TRACK

Upon receipt of the READ track command, the head is loaded, and the Busy Status bit is set. Reading starts with the leading edge of the first encountered index pulse and continues until the next index pulse. All Gap, Header, and data bytes are assembled and transferred to the data register and DRQ's are generated for each byte. The accumulation of bytes is synchronized to each address mark encountered. An interrupt is generated at the completion of the command.

This command has several characteristics which make it suitable for diagnostic purposes. They are: the Read Gate

is not activated during the command; no CRC checking is performed; gap information is included in the data stream; the internal side compare is not performed; and the address mark detector is on for the duration of the command. Because the A.M. detector is always on, write splices or noise may cause the chip to look for an A.M. If an address mark does not appear on schedule the Lost Data status flag is set.

The ID A.M., ID field, ID CRC bytes, DAM, Data, and Data CRC Bytes for each sector will be correct. The Gap Bytes may be read incorrectly during write-splice time because of synchronization.

**TYPE III COMMAND WRITE TRACK****TYPE III COMMAND WRITE TRACK**

FD179X-02

CONTROL BYTES FOR INITIALIZATION

DATA PATTERN IN DR (HEX)	FD179X INTERPRETATION IN FM (DDEN = 1)	FD1791/3 INTERPRETATION IN MFM (DDEN = 0)
00 thru F4	Write 00 thru F4 with CLK = FF	Write 00 thru F4, in MFM
F5	Not Allowed	Write A1* in MFM, Preset CRC
F6	Not Allowed	Write C2** in MFM
F7	Generate 2 CRC bytes	Generate 2 CRC bytes
F8 thru FB	Write F8 thru FB, Clk = C7, Preset CRC	Write F8 thru FB, in MFM
FC	Write FC with Clk = D7	Write FC in MFM
FD	Write FD with Clk = FF	Write FD in MFM
FE	Write FE, Clk = C7, Preset CRC	Write FE in MFM
FF	Write FF with Clk = FF	Write FF in MFM

*Missing clock transition between bits 4 and 5

**Missing clock transition between bits 3 & 4

WRITE TRACK FORMATTING THE DISK

(Refer to section on Type III commands for flow diagrams.)

Formatting the disk is a relatively simple task when operating programmed I/O or when operating under DMA with a large amount of memory. Data and gap information must be provided at the computer interface. Formatting the disk is accomplished by positioning the R/W head over the desired track number and issuing the Write Track command.

Upon receipt of the Write Track command, the head is loaded and the Busy Status bit is set. Writing starts with the leading edge of the first encountered Index pulse and continues until the next index pulse, at which time the interrupt is activated. The Data Request is activated immediately upon receiving the command, but writing will not start until after the first byte has been loaded into the Data Register. If the DR has not been loaded by the time the index pulse is encountered the operation is terminated making the device Not Busy, the Lost Data Status Bit is set, and the interrupt is activated. If a byte is not present in the DR when needed, a byte of zeroes is substituted.

This sequence continues from one Index mark to the next index mark. Normally, whatever data pattern appears in the data register is written on the disk with a normal clock pattern. However, if the FD179X detects a data pattern of F5 thru FE in the data register, this is interpreted as data address marks with missing clocks or CRC generation.

The CRC generator is initialized when any data byte from F8 to FE is about to be transferred from the DR to the DSR in FM or by receipt of F5 in MFM. An F7 pattern will generate two CRC characters in FM or MFM. As a consequence, the patterns F5 thru FE must not appear in the gaps, data fields, or ID fields. Also, CRC's must be generated by an F7 pattern.

Disks may be formatted in IBM 3740 or System 34 formats with sector lengths of 128, 256, 512, or 1024 bytes.

TYPE IV COMMANDS

The Forced Interrupt command is generally used to terminate a multiple sector read or write command or to in-

sure Type I status in the status register. This command can be loaded into the command register at any time. If there is a current command under execution (busy status bit set) the command will be terminated and the busy status bit reset.

The lower four bits of the command determine the conditional interrupt as follows:

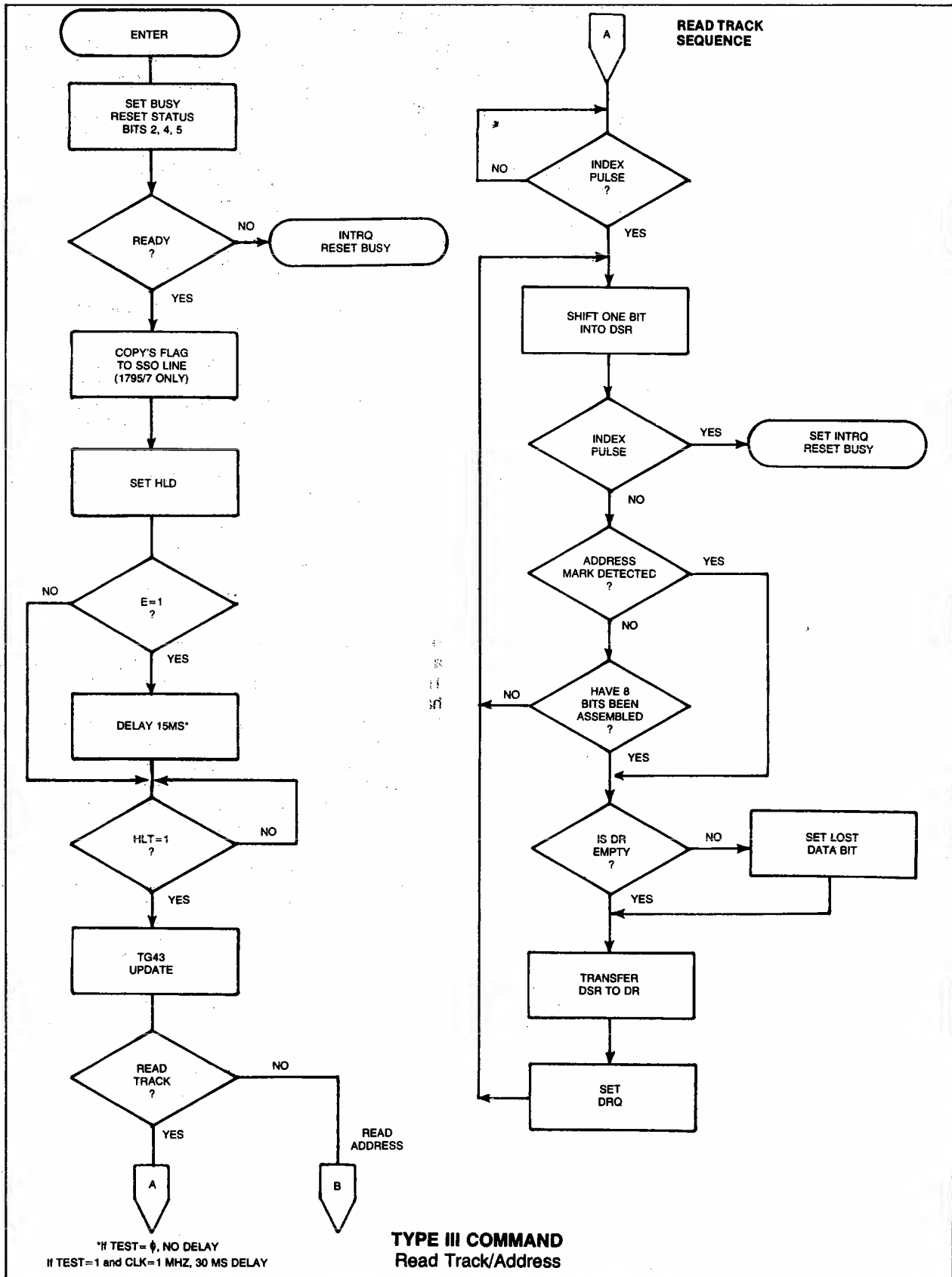
- I₀ = Not-Ready to Ready Transition
- I₁ = Ready to Not-Ready Transition
- I₂ = Every Index Pulse
- I₃ = Immediate Interrupt

The conditional interrupt is enabled when the corresponding bit positions of the command (I₃ - I₀) are set to a 1. Then, when the condition for interrupt is met, the INTRQ line will go high signifying that the condition specified has occurred. If I₃ - I₀ are all set to zero (HEX D0), no interrupt will occur but any command presently under execution will be immediately terminated. When using the immediate interrupt condition (I₃ = 1) an interrupt will be immediately generated and the current command terminated. Reading the status or writing to the command register will not automatically clear the interrupt. The HEX D0 is the only command that will enable the immediate interrupt (HEX D8) to clear on a subsequent load command register or read status register operation. Follow a HEX D8 with D0 command.

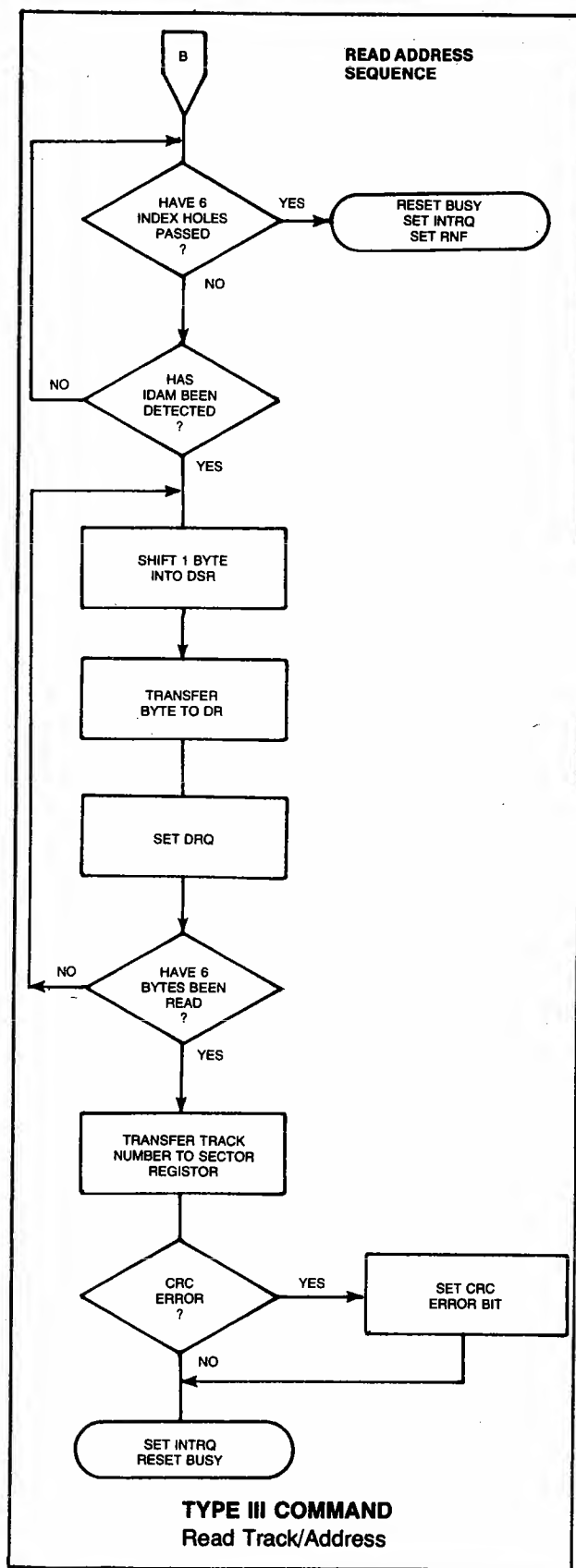
Wait 8 micro sec (double density) or 16 micro sec (single density) before issuing a new command after issuing a forced interrupt (times double when clock = 1 MHz). Loading a new command sooner than this will nullify the forced interrupt.

Forced Interrupt stops any command at the end of an internal micro-instruction and generates INTRQ when the specified condition is met. Forced interrupt will wait until ALU operations in progress are complete (CRC calculations, compares, etc.).

More than one condition may be set at a time. If for example, the READY TO NOT-READY condition (I₁ = 1) and the Every Index Pulse (I₂ = 1) are both set, the resultant command would be HEX "DA". The "OR" function is performed so that either a READY TO NOT-READY or the next Index Pulse will cause an interrupt condition.



FD179X-02

**STATUS REGISTER**

Upon receipt of any command, except the Force Interrupt command, the Busy Status bit is set and the rest of the status bits are updated or cleared for the new command. If the Force Interrupt Command is received when there is a current command under execution, the Busy status bit is reset, and the rest of the status bits are unchanged. If the Force Interrupt command is received when there is not a current command under execution, the Busy Status bit is reset and the rest of the status bits are updated or cleared. In this case, Status reflects the Type I commands.

The user has the option of reading the status register through program control or using the DRQ line with DMA or interrupt methods. When the Data register is read the DRQ bit in the status register and the DRQ line are automatically reset. A write to the Data register also causes both DRQ's to reset.

The busy bit in the status may be monitored with a user program to determine when a command is complete, in lieu of using the INTRQ line. When using the INTRQ, a busy status check is not recommended because a read of the status register to determine the condition of busy will reset the INTRQ line.

The format of the Status Register is shown below:

(BITS)							
7	6	5	4	3	2	1	0
S7	S6	S5	S4	S3	S2	S1	S0

Status varies according to the type of command executed as shown in Table 4.

Because of internal sync cycles, certain time delays must be observed when operating under programmed I/O. They are: (times double when clock = 1 MHz)

Operation	Next Operation	Delay Req'd.	
		FM	MFM
Write to Command Reg.	Read Busy Bit (Status Bit 0)	12 μ s	6 μ s
Write to Command Reg.	Read Status Bits 1-7	28 μ s	14 μ s
Write Any Register	Read From Diff. Register	0	0

IBM 3740 FORMAT — 128 BYTES/SECTOR

Shown below is the IBM single-density format with 128 bytes/sector. In order to format a diskette, the user must issue the Write Track command, and load the data register with the following values. For every byte to be written, there is one Data Request.

IBM SYSTEM 34 FORMAT. 256 BYTES/SECTOR

Shown below is the IBM dual-density format with 256 bytes/sector. In order to format a diskette the user must issue the Write Track command and load the data register with the following values. For every byte to be written, there is one data request.

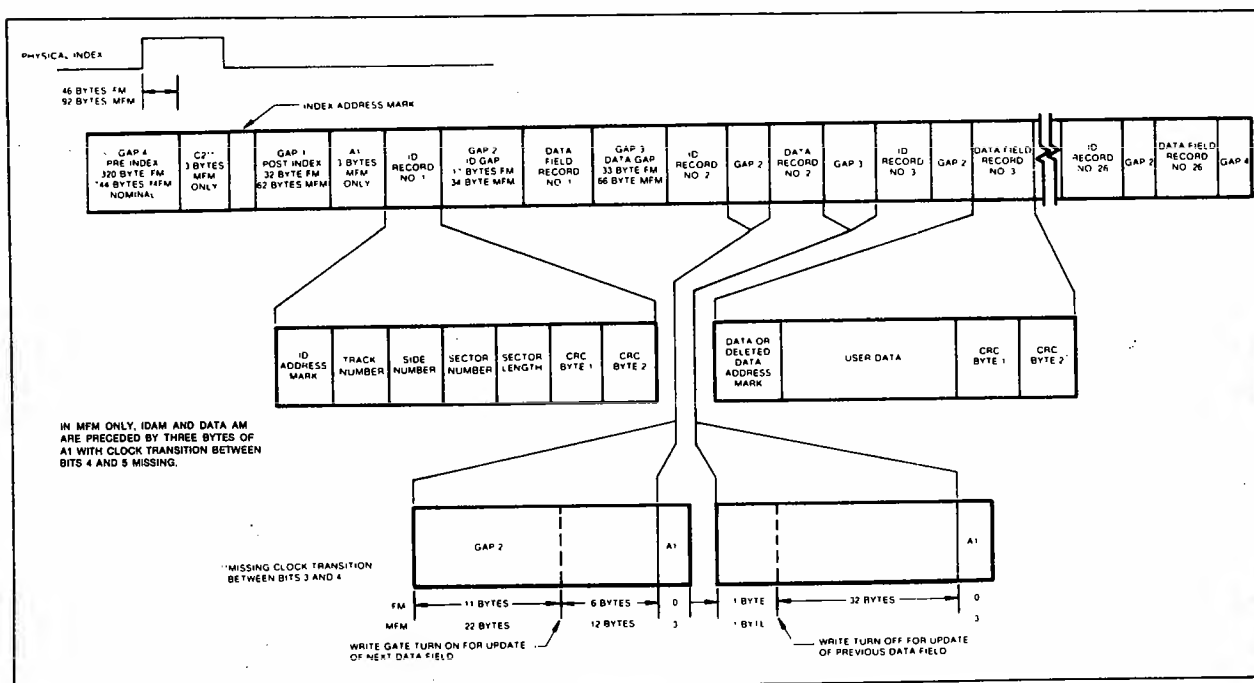
NUMBER OF BYTES	HEX VALUE OF BYTE WRITTEN
80	4E
12	00
3	F6 (Writes C2)
1	FC (Index Mark)
* 50	4E
12	00
3	F5 (Writes A1)
1	FE (ID Address Mark)
1	Track Number (0 thru 4C)
1	Side Number (0 or 1)
1	Sector Number (1 thru 1A)
1	01 (Sector Length)
1	F7 (2 CRCs written)
22	4E
12	00
3	F5 (Writes A1)
1	FB (Data Address Mark)
256	DATA
1	F7 (2 CRCs written)
54	4E
598**	4E

*Write bracketed field 26 times

****Continue writing until FD179X interrupts out.**

Approx. 598 bytes.

1-Optional '00' on 1795/7 only.



IBM TRACK FORMAT

1. NON-IBM FORMATS

Variations in the IBM formats are possible to a limited extent if the following requirements are met:

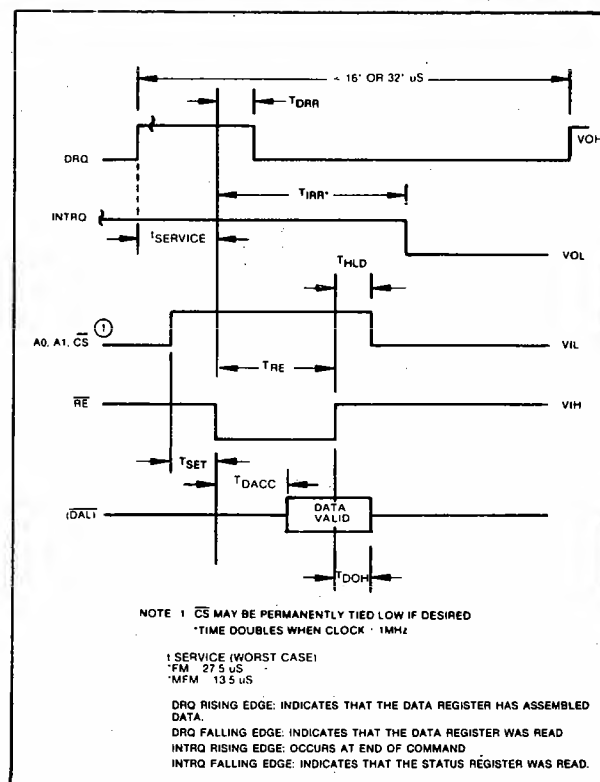
- 1) Sector size must be 128, 256, 512 or 1024 bytes.
- 2) Gap 2 cannot be varied from the IBM format.
- 3) 3 bytes of A1 must be used in MFM.

In addition, the Index Address Mark is not required for operation by the FD179X. Gap 1, 3, and 4 lengths can be as short as 2 bytes for FD179X operation, however PLL lock up time, motor speed variation, write-splice area, etc. will add more bytes to each gap to achieve proper operation. It is recommended that the IBM format be used for highest system reliability.

	FM	MFM
Gap I	16 bytes FF	32 bytes 4E
Gap II	11 bytes FF	22 bytes 4E
*	6 bytes 00	12 bytes 00
*		3 bytes A1
Gap III**	10 bytes FF 4 bytes 00	24 bytes 4E 8 bytes 00 3 bytes A1
Gap IV	16 bytes FF	16 bytes 4E

*Byte counts must be exact.

**Byte counts are minimum, except exactly 3 bytes of A1 must be written.



READ ENABLE TIMING

TIMING CHARACTERISTICS

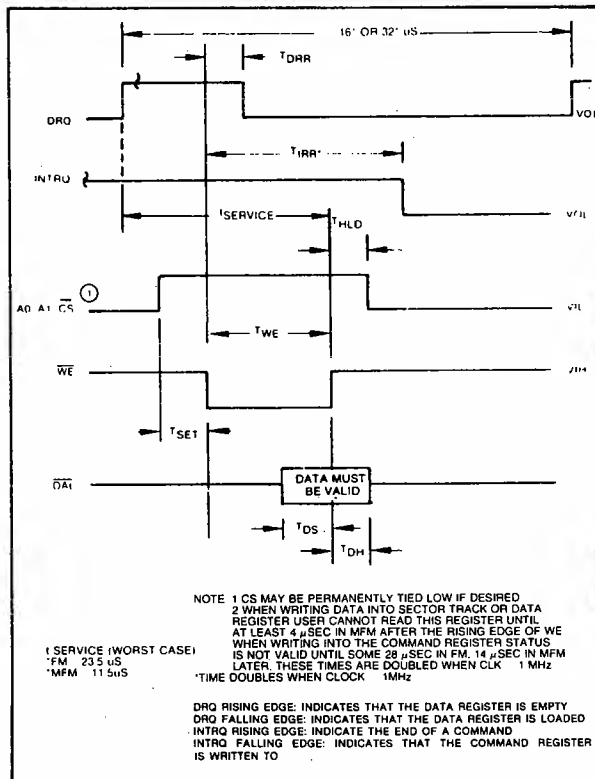
$T_A = 0^\circ\text{C}$ to 70°C , $V_{DD} = +12\text{V} \pm .6\text{V}$, $V_{SS} = 0\text{V}$, $V_{CC} = +5\text{V} \pm .25\text{V}$

READ ENABLE TIMING (See Note 6, Page 21)

SYMBOL	CHARACTERISTIC	MIN.	TYP.	MAX.	UNITS	CONDITIONS
TSET	Setup ADDR & CS to $\overline{\text{RE}}$	50			nsec	
THLD	Hold ADDR & CS from $\overline{\text{RE}}$	10			nsec	
TRE	$\overline{\text{RE}}$ Pulse Width	400			nsec	$C_L = 50 \text{ pf}$
TDRR	DRQ Reset from $\overline{\text{RE}}$		400	500	nsec	
TIRR	INTRQ Reset from $\overline{\text{RE}}$		500	3000	nsec	See Note 5
TDACC	Data Access from $\overline{\text{RE}}$			350	nsec	$C_L = 50 \text{ pf}$
TDOH	Data Hold From $\overline{\text{RE}}$	50		150	nsec	$C_L = 50 \text{ pf}$

WRITE ENABLE TIMING (See Note 6, Page 21)

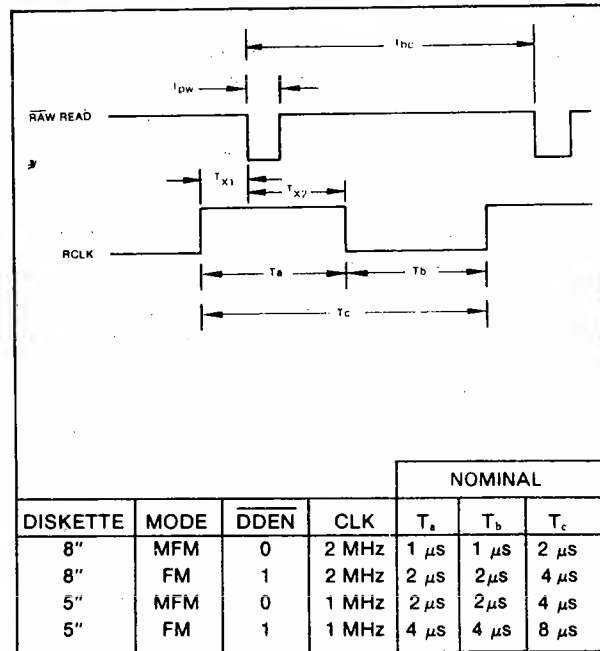
SYMBOL	CHARACTERISTIC	MIN.	TYP.	MAX.	UNITS	CONDITIONS
TSET	Setup ADDR & CS to $\overline{\text{WE}}$	50			nsec	
THLD	Hold ADDR & CS from $\overline{\text{WE}}$	10			nsec	
TWE	$\overline{\text{WE}}$ Pulse Width	350			nsec	
TDRR	DRQ Reset from $\overline{\text{WE}}$		400	500	nsec	
TIRR	INTRQ Reset from $\overline{\text{WE}}$		500	3000	nsec	See Note 5
TDS	Data Setup to $\overline{\text{WE}}$	250			nsec	
TDH	Data Hold from $\overline{\text{WE}}$	70			nsec	

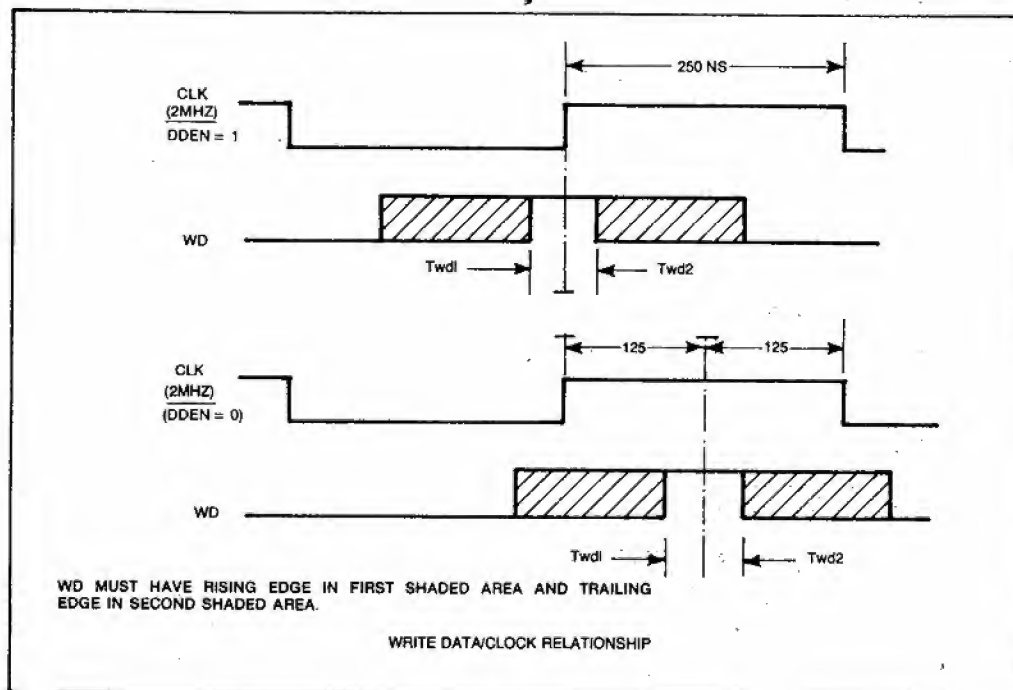
**WRITE ENABLE TIMING****INPUT DATA TIMING:**

SYMBOL	CHARACTERISTIC	MIN.	TYP.	MAX.	UNITS	CONDITIONS
Tpw	Raw Read Pulse Width	100	200		nsec	See Note 1
tbc	Raw Read Cycle Time	1500	2000		nsec	1800 ns @ 70°C
Tc	RCLK Cycle Time	1500	2000		nsec	1800 ns @ 70°C
Tx ₁	RCLK hold to Raw Read	40			nsec	See Note 1
Tx ₂	Raw Read hold to RCLK	40			nsec	See Note 1

WRITE DATA TIMING: (ALL TIMES DOUBLE WHEN CLK = 1 MHz) (See Note 6, Page 21)

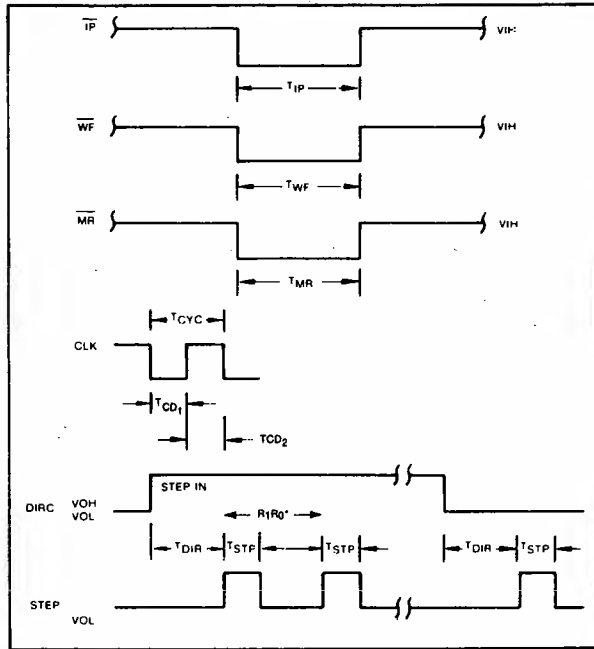
SYMBOL	CHARACTERISTICS	MIN.	TYP.	MAX.	UNITS	CONDITIONS
Twp	Write Data Pulse Width		500 200	650 350	nsec	FM
Twg	Write Gate to Write Data		2 1		nsec μ sec	MFM
Tbc	Write data cycle Time		2,3, or 4		μ sec	FM
Ts	Early (Late) to Write Data	125			nsec	±CLK Error
Th	Early (Late) From Write Data	125			nsec	MFM
Twf	Write Gate off from WD		2 1		μ sec μ sec	FM
Twd1	WD Valid to Clk	100			nsec	MFM
Twd2	WD Valid after CLK	50 100 30			nsec nsec nsec	CLK=1 MHZ CLK=2 MHZ CLK=1 MHZ CLK=2 MHZ

**INPUT DATA TIMING**

**WRITE DATA TIMING**

MISCELLANEOUS TIMING: (Times Double When Clock = 1 MHz) (See Note 6, Page 21)

SYMBOL	CHARACTERISTIC	MIN.	TYP.	MAX.	UNITS	CONDITIONS
TCD ₁	Clock Duty (low)	230	250	20000	nsec	See Note 5 ± CLK ERROR
TCD ₂	Clock Duty (high)	200	250	20000	nsec	
TSTP	Step Pulse Output	2 or 4			μsec	
TDIR	Dir Setup to Step		12		μsec	
TMR	Master Reset Pulse Width	50			μsec	See Note 5
TIP	Index Pulse Width	10			μsec	
TWF	Write Fault Pulse Width	10			μsec	

**MISCELLANEOUS TIMING**

*FROM STEP RATE TABLE

Table 4. STATUS REGISTER SUMMARY

BIT	ALL TYPE I COMMANDS	READ ADDRESS	READ SECTOR	READ TRACK	WRITE SECTOR	WRITE TRACK
S7	NOT READY	NOT READY	NOT READY	NOT READY	NOT READY	NOT READY
S6	WRITE PROTECT	0	0	0	WRITE PROTECT	WRITE PROTECT
S5	HEAD LOADED	0	RECORD TYPE	0	WRITE FAULT	WRITE FAULT
S4	SEEK ERROR	RNF	RNF	0	RNF	0
S3	CRC ERROR	CRC ERROR	CRC ERROR	0	CRC ERROR	0
S2	TRACK 0	LOST DATA	LOST DATA	LOST DATA	LOST DATA	LOST DATA
S1	INDEX PULSE*	DRQ	DRQ	DRQ	DRQ	DRQ
S0	BUSY	BUSY	BUSY	BUSY	BUSY	BUSY

STATUS FOR TYPE I COMMANDS

BIT NAME	MEANING
S7 NOT READY	This bit when set indicates the drive is not ready. When reset it indicates that the drive is ready. This bit is an inverted copy of the Ready input and logically 'ored' with MR.
S6 PROTECTED	When set, indicates Write Protect is activated. This bit is an inverted copy of WRPT input.
S5 HEAD LOADED	When set, it indicates the head is loaded and engaged. This bit is a logical "and" of HLD and HLT signals.
S4 SEEK ERROR	When set, the desired track was not verified. This bit is reset to 0 when updated.
S3 CRC ERROR	CRC encountered in ID field.
S2 TRACK 00	When set, indicates Read/Write head is positioned to Track 0. This bit is an inverted copy of the TROO input.
S1 INDEX	When set, indicates index mark detected from drive. This bit is an inverted copy of the IP input.
S0 BUSY	When set command is in progress. When reset no command is in progress.

NOTES:

1. Pulse width on RAW READ (Pin 27) is normally 100-300 ns. However, pulse may be any width if pulse is entirely within window. If pulse occurs in both windows, then pulse width must be less than 300 ns for MFM at CLK = 2 MHz and 600 ns for FM at 2 MHz. Times double for 1 MHz.
2. A PPL Data Separator is recommended for 8" MFM.
3. tbc should be 2 μ s, nominal in MFM and 4 μ s nominal in FM. Times double when CLK = 1 MHz.
4. RCLK may be high or low during RAW READ (Polarity is unimportant).
5. Times double when clock = 1 MHz.
6. Output timing readings are at $V_{OL} = 0.8v$ and $V_{OH} = 2.0v$.

FD179X-02

STATUS FOR TYPE II AND III COMMANDS

BIT NAME	MEANING
S7 NOT READY	This bit when set indicates the drive is not ready. When reset, it indicates that the drive is ready. This bit is an inverted copy of the Ready input and 'ored' with MR. The Type II and III Commands will not execute unless the drive is ready.
S6 WRITE PROTECT	On Read Record: Not Used. On Read Track: Not Used. On any Write: It indicates a Write Protect. This bit is reset when updated.
S5 RECORD TYPE/ WRITE FAULT	On Read Record: It indicates the record-type code from data field address mark. 1 = Deleted Data Mark. 0 = Data Mark. On any Write: It indicates a Write Fault. This bit is reset when updated.
S4 RECORD NOT FOUND (RNF)	When set, it indicates that the desired track, sector, or side were not found. This bit is reset when updated.
S3 CRC ERROR	If S4 is set, an error is found in one or more ID fields; otherwise it indicates error in data field. This bit is reset when updated.
S2 LOST DATA	When set, it indicates the computer did not respond to DRQ in one byte time. This bit is reset to zero when updated.
S1 DATA REQUEST	This bit is a copy of the DRQ output. When set, it indicates the DR is full on a Read Operation or the DR is empty on a Write operation. This bit is reset to zero when updated.
S0 BUSY	When set, command is under execution. When reset, no command is under execution.

ELECTRICAL CHARACTERISTICS**Absolute Maximum Ratings**V_{DD} with respect to V_{SS} (ground): + 15 to - 0.3VVoltage to any input with respect to V_{SS} = + 15 to - 0.3VI_{CC} = 60 mA (35 mA nominal)I_{DD} = 15 mA (10 mA nominal)C_{IN} & C_{OUT} = 15 pF max with all pins grounded except one under test.

Operating temperature = 0°C to 70°C

Storage temperature = - 55°C to + 125°C

OPERATING CHARACTERISTICS (DC)TA = 0°C to 70°C, V_{DD} = + 12V ± .6V, V_{SS} = 0V, V_{CC} = + 5V ± .25V

SYMBOL	CHARACTERISTIC	MIN.	MAX.	UNITS	CONDITIONS
I _{IL}	Input Leakage		10	μA	V _{IN} = V _{DD} **
I _{OL}	Output Leakage		10	μA	V _{OUT} = V _{DD}
V _{IH}	Input High Voltage	2.6		V	
V _{IL}	Input Low Voltage		0.8	V	
V _{OH}	Output High Voltage	2.8		V	I _O = - 100 μA
V _{OL}	Output Low Voltage		0.45	V	I _O = 1.6 mA*
P _D	Power Dissipation		0.6	W	

*1792 and 1794 I_O = 1.0 mA

**Leakage conditions are for input pins without internal pull-up resistors. Pins 22, 23, 33, 36, and 37 have pull-up resistors. See Tech Memo #115 for testing procedures.

See page 725 for ordering information.

WESTERN DIGITAL

C O R P O R A T I O N

FD179X Application Notes

FD179X

INTRODUCTION

Over the past several years, the Floppy Disk Drive has become the most popular on-line storage device for mini and microcomputer systems. Its fast access time, reliability and low cost-per-bit ratio enables the Floppy Disk Drive to be the solution in mass storage for microprocessor systems. The drive interface to the Host system is standardized, allowing the OEM to substitute one drive for another with minimum hardware/ software modifications.

Since Floppy Disk Data is stored and retrieved as a self-clocking serial data stream, some means of separating the clock from the data and assembling this data in parallel form must be accomplished. Data is stored on individual Tracks of the media, requiring control of a stepper motor to move the Read/Write head to a predetermined Track. Byte synchronization must also be accomplished to insure that the parallel data is properly assembled. After all the design considerations are met, the final controller can consist of 40 or more TTL packages.

To alleviate the burden of Floppy Disk Controller design, Western Digital has developed a Family of LSI Floppy Disk controller devices. Through its own set of macro commands, the FD179X Controller Family will perform all the functions necessary to read and write data to the drive. Both the 8" standard and 5¼" mini-floppy are supported with single or double density recording techniques. The FD179X is compatible with the IBM 3740 (FM) data format, or the System 34 (MFM) standards. Provisions for non-standard formats and variable sector lengths have been included to provide more storage capability per track. Requiring standard +5, +12 power supplies the FD179X is available in a standard 40 pin dual-in-line package.

The FD179X Family consists of 6 devices. The differences between these devices is summarized in Figure 1. The 1792 and 1794 are "single density only" devices, with the Double Density Enable pin (DDEN) left open by the user. Both True and inverted Data bus devices are available. Since the 179X can only drive one TTL Load, a true data bus system may use the 1791 with external inverting buffers to arrive at a true bus scheme. The 1795 and 1797 are identical to the 1791 and 1793, except a side select output has been added that is controlled through the Command Register.

SYSTEM DESIGN

The first consideration in Floppy Disk Design is to determine which type of drive to use. The choice ranges from single-density single sided mini-floppy to the 8" double-density double-sided drive. Figure 2 illustrates the various drive and data capacities associated with each type. Although the 8" double-density drive offers twice as much storage, a more complex data separator and the addition of Write Precompensation circuits are mandatory for reliable data transfers. Whether to go with 8" double-density or not is dependent upon PC board space and the additional circuitry needed to accurately recover data with extreme bit shifts. The byte transfer time defines the nominal time required to transfer one byte of data from the drive. If the CPU used cannot service a byte in this time, then a DMA scheme will probably be required. The 179X also needs a few microseconds for overhead, which is subtracted from the transfer time. Figure 3 shows the actual service times that the CPU must provide on a byte-by-byte basis. If these times are not met, bytes of data will be lost during a read or write operation. For each byte transferred, the 179X generates a DRQ (Data Request) signal on Pin 38. A bit is provided in the status register which is also set upon receipt of a byte from the Disk. The user has the option of reading the status register through program control or using the DRQ Line with DMA or interrupt schemes. When the data register is read, both the status register DRQ bit and the DRQ Line are automatically reset. The next full byte will again set the DRQ and the process continues until the sector(s) are read. The Write operation works exactly the same way, except a WRITE to the Data Register causes a reset of both DRQ's.

RECORDING FORMATS

The FD179X accepts data from the disk in a Frequency-Modulated (FM) or Modified-Frequency-Modulated (MFM) Format. Shown in Figures 4A and 4B are both these Formats when writing a Hexidecimal byte of 'D2'. In the FM mode, the 8 bits of data are broken up into "bit cells." Each bit cell begins with a clock pulse and the center of the bit cell defines the data. If the data bit = 0, no pulse is written; if the data = 1, a pulse is written in the center of the cell. For the 8" drive, each clock is written 4 microseconds apart.

In the MFM mode, clocks are decoded into the data stream. The byte is again broken up into bit cells, with the data bit written in the center of the bit cell if data = 1. Clocks are only written if both surrounding data bits are zero. Figure 4B shows that this occurs only once between Bit cell 4 and 5. Using this encoding scheme, pulses can occur 2, 3 or 4 microseconds apart. The bit cell time is now 2 microseconds; twice as much data can be recorded without increasing the Frequency rate due to this encoding scheme.

The 179X was designed to be compatible with the IBM 3740 (FM) and System 34 (MFM) Formats. Although most users do not have a need for data exchange with IBM mainframes, taking advantage of these well studied formats will insure a high degree of system performance. The 179X will allow a change in gap fields and sector lengths to increase usable storage capacity, but variations away from these standards is not recommended. Both IBM standards are soft-sector format. Because of the wide variation in address marks, the 179X can only support soft-sectored media. Hard sector diskettes have continued to lose popularity, mainly due to the unavailability of a standard and the limitation of sector lengths imposed by the physical sector holes in the diskette.

PROCESSOR INTERFACE

The Interface of the 179X to the CPU consists of an 8-bit Bi-directional bus, read/write controls and optional interrupt lines. By selecting the device via the CHIP SELECT Line, each of the five internal registers can be accessed.

Shown below are the registers and their addresses:

PIN 3 CS	PIN 6 A ₁	PIN 5 A ₀	PIN 4 RE=0	PIN 2 WE=0
0	0	0	STATUS REG	COMMAND REG
0	0	1	TRACK REG	REG
0	1	0	SECTOR REG	TRACK REG
0	1	1	DATA REG	SECTOR REG
1	X	X	H1-Z	DATA REG H1-Z

Each time a command is issued to the 179X, the Busy bit is set and the INTRQ (Interrupt Request) Line is reset. The user has the option of checking the busy bit or use the INTRQ Line to denote command completion. The Busy bit will be reset whenever the 179X is idle and awaiting a new command. The INTRQ Line, once set, can only be reset by a READ of the status register or issuing a new command. The MR (Master Reset) Line does not affect INTRQ.

The A₀, A₁ Lines used for register selections can be configured at the CPU in a variety of ways. These lines may actually tie to CPU address lines, in which case the 179X will be memory-mapped and addressed like RAM. They may also be used under Program Control by tying to a port device such as the 8255, 6820, etc. As a diagnostic tool when checking out the CPU interface, the Track and Sector registers should respond like "RAM" when the 179X is idle (Busy = INTRQ = 0).

Because of internal synchronization cycles, certain time delays must be introduced when operating under Programmed I/O. The worst case delays are:

OPERATION	NEXT OPERATION	DELAY REQ'D
WRITE TO COMMAND REG	READ STATUS REGISTER	MFM = 14μs* FM = 28μs.
WRITE TO ANY REGISTER	READ FROM A DIFFERENT REG	NO DELAY

*NOTE: Times Double when CLK = 1MHz (5¼" drive)

Other CPU interface lines are CLK, \overline{MR} and \overline{DDEN} . The CLK line should be 2MHz (8" drive) or 1MHz (5¼" drive) with a 50% duty cycle. Accuracy should be $\pm 1\%$ (crystal source) since all internal timing, including stepping rates, are based upon this clock.

The \overline{MR} or Master Reset Line should be strobed a minimum of 50 microseconds upon each power-on condition. This line clears and initializes all internal registers and issues a restore command (Hex '03') on the rising edge. A quicker stepping rate can be written to the command register after a \overline{MR} , in which case the remaining steps will occur at the faster programmed rate. The 179X will issue a maximum of 255 stepping pulses in an attempt to expect the TROO line to go active low. This line should be connected to the drive's TROO sensor.

The \overline{DDEN} line causes selection of either single density ($\overline{DDEN} = 1$) or double density operation. \overline{DDEN} should not be switched during a read or write operation.

FLOPPY DISK INTERFACE

The Floppy Disk Interface can be divided into three sections: Motor Control, Write Signals and Read Signals. All of these lines are capable of driving one TTL load and not compatible for direct connection to the drive. Most drives require an open-collector TTL interface with high current drive capability. This must be done on all outputs from the 179X. Inputs to the 179X may be buffered or tied to the Drives outputs, providing the appropriate resistor termination networks are used. Undershoot should not exceed -0.3 volts, while integrity of V_{IH} and V_{OH} levels should be kept within spec.

MOTOR CONTROL

Motor Control is accomplished by the STEP and DIRC Lines. The STEP Line issues stepping pulses with a period defined by the rate field in all Type I commands. The DIRC Line defines the direction of steps (DIRC = 1 STEP IN/DIRC = 0 STEP OUT).

Other Control Lines include the \overline{IP} or Index Pulse. This Line is tied to the drives' Index L.E.D. sensor and makes an active transition for each revolution of the diskette. The TROO Line is another L.E.D. sensor that informs the 179X that the stepper motor is at its furthest position, over Track 00. The READY Line can be used for a number of functions, such as sensing "door open", Drive motor on, etc. Most drives provide a programmable READY Signal selected by option jumpers on the drive. The 179X will look at the ready signal prior to executing READ/WRITE commands. READY is *not* inspected during any Type I commands. All Type I commands will execute regardless of the Logic Level on this Line.

WRITE SIGNALS

Writing of data is accomplished by the use of the WD, WG, WF, TG43, EARLY and LATE Lines. The WG or Write Gate Line is used to enable write current at the drive's R/W head. It is made active prior to writing data on the disk. The WF or WRITE FAULT Line is used to inform the 179X of a failure in drive electronics. This signal is multiplexed with the VFOE Line and must be logically separated if required. Figure 5 illustrates three methods of demultiplexing.

The TG43 or "TRACK GREATER than 43" Line is used to decrease the Write current on the inner tracks, where bit densities are the highest. If not required on the drive, TG43 may be left open.

WRITE PRECOMPENSATION

The 179X provides three signals for double density Write Precompensation use. These signals are WRITE DATA, EARLY and LATE. When using single density drives (either 8" or 5¼"), Write Precompensation is not necessary and the WRITE DATA line is generally TTL Buffered and sent directly to the drive. In this mode, EARLY and LATE are left open.

For double density use, Write Precompensation is a function of the drive. Some manufacturers recommend Precompensating the 5¼" drive, while others do not.

With the 8" drive, Precompensation may be specified from TRACK 43 on, or in most cases, all TRACKS. If the recommended Precompensation is not specified, check with the manufacturer for the proper configuration required.

The amount of Precompensation time also varies. A typical value will usually be specified from 100-300ns. Regardless of the parameters used, Write Precompensation must be done external to the 179X. When \overline{DDEN} is tied low, EARLY or LATE will be activated at least 125ns. before and after the Write Data pulse. An Algorithm internal the 179X decides whether to raise EARLY or LATE, depending upon the previous bit pattern sent. As an example, suppose the recommended Precomp value has been specified at 150ns. The following action should be taken:

EARLY	LATE	ACTION TAKEN
0	0	delay WD by 150ns (nominal)
0	1	delay WD by 300ns (2X value)
1	0	do not delay WD

There are two methods of performing Write Precompensation:

- 1) External Delay elements
- 2) Digitally

Shown in Figure 6 is a Precomp circuit using the Western Digital 2143 clock generator as the delay element. The WD pulse from the 179X creates a strobe to the 2143, causing subsequent output pulses on the $\phi 1$, $\phi 2$ and $\phi 3$ signals. The 5K Precomp adjust sets the desired Precomp value. Depending upon the condition of EARLY and LATE, $\phi 1$ will be used for EARLY, $\phi 2$ for nominal (EARLY = LATE = 0), and $\phi 3$ for LATE. The use of "one-shots" or delay line in a Write Precompensation scheme offers the user the ability to vary the Precomp value. The $\phi 4$ output resets the 74LS175 Latch in anticipation of the next WD pulse. Figure 7 shows the WD-EARLY/LATE relationship, while Figure 8 shows the timing of this write Precomp scheme.

Another method of Precomp is to perform the function digitally. Figure 9 illustrates a relationship between the WD pulse and the CLK pin, allowing a digital Precomp scheme. Figure 10 shows such a scheme with a preset Write Precompensation value of 250ns. The synchronous counter is used to generate 2MHz and 4MHz clock signals. The 2MHz clock is sent to the CLK input of the 179X and the 4MHz is used by the 4-bit shift register. When a WD pulse is not present, the 4MHz clock is shifting "ones" through the shift register and maintaining Q_D at a zero level. When a WD pulse is present, a zero is loaded at either A, B, or C depending upon the states of LATE, EN PRECOMP and EARLY. The zero is then shifted by the 4MHz clock until it reaches the Q_D output. The number of shift operations determines whether the WRITE DATA pulse is written early, nominal or late. If both FM and MFM operations is a system requirement, the output of this circuit should be disabled and the WD pulse should be sent directly to the drive.

DATA SEPARATION

The 179X has two inputs (RAW READ & RCLK) and one output (VFOE) for use by an external data separator. The RAW READ input must present clock and data pulses to the 179X, while the RCLK input provides a "window" or strobe signal to clock each RAW READ pulse into the device. An ideal Data Separator would have the leading edge of the RAW READ pulse occur in the exact center of the RCLK strobe.

Motor Speed Variation, Bit shifts and read amplifier recovery circuits all cause the RAW READ pulses to drift away from their nominal positions. As this occurs, the RAW READ pulses will shift left or right with respect to RCLK. Eventually, a pulse will make its transition outside of its RCLK window, causing either a CRC error or a Record-not-Found error at the 179X.

A Phase-Lock-Loop circuit is one method of achieving synchronization between the RCLK and RAW READ signals. As RAW READ pulses are fed to the PLL, minor adjustments of the free-running RCLK frequency can be made. If pulses are occurring too far apart, the RCLK frequency is *decreased* to keep synchronization. If pulses begin to occur closer together, RCLK is *increased* until this new higher frequency is achieved. In normal read operations, RCLK will be constantly adjusted in an attempt to match the incoming RAW READ frequency.

Another method of Data Separation is the Counter-Separator technique. The RCLK signal is again free-running at a nominal rate, until a RAW READ pulse occurs. The Separator then denotes the position of the pulse with respect to RCLK (by the counter value), and counts down to increase or decrease the current RCLK window. The next RCLK window will occur at a nominal rate and will continue to run at this frequency until another RAW READ pulse adjusts RCLK, but only the present window is adjusted.

Both PPL and Counter/Separator are acceptable methods of Data Separation. The PPL has the highest reliability because of its "tracking" capability and is recommended for 8" double density designs.

As a final note, the term "Data Separator" may be misleading, since the physical separation of clock and data bits are not actually performed. This term is used throughout the industry, and can better be described as a "Data Recovery Circuit" rather than a Data Separator.

The VFOE signal is an output from the 179X that signifies the head has been loaded and valid data pulses are appearing on the RAW READ line. It can be used to enable the Data Separator and to insure clean RCLK transitions to the 179X. Since some drives will output random pulses when the head is disengaged, VFOE can prevent an erratic RCLK signal during this time. If the Data Separator requires synchronization during a known pattern of one's or zero's, then RG (READ GATE) can be used. The RG signal will go active when the 179X is currently over a field of zeros or ones. RG is not available on the 1795/1797 devices, since this signal was replaced with the SSO (Side Select Output) Line.

Shown in Figure 11 is a $2\frac{1}{2}$ IC Counter/Separator. The 74LS193 free runs at a frequency determined by the CRYCLK input. When a RAW READ pulse occurs, the counter is loaded with a starting count of '5'. When the RAW READ Line returns to a Logic 1, the counter counts down to zero and again free runs. The 74LS74 insures a 50% duty cycle to the 179X and performs a divide-by-two of the Q_D output.

Figure 12 illustrates another Counter/Separator utilizing a PROM as the count generator. Depending upon the RAW READ phase relationship to RCLK, the PROM is addressed and its data output is used as the counter value. A 16MHz clock is required for 8" double density, while an 8MHz clock can be used for single density.

Figure 13 shows a Phase-Lock-Loop data recovery circuit. The phase detector (U2, Figure 2) compares the phase of the SHAPED DATA pulse to the phase of $VFO\ CLK \div 2$. If $VFO\ CLK \div 2$ is lagging the SHAPED DATA pulse an output pulse on #9, U2 is generated. The filter/amplifier converts this pulse into a DC signal which increases the frequency of the VCO.

If, correspondingly, $CLK \div 2$ is leading the SHAPED DATA pulse, an output pulse on #5, U2 is generated. This pulse is converted into a DC signal which decreases the frequency of the VCO. These two actions cause the VCO to track the frequency of the incoming READ DATA pulses. This correction process to keep the two signals in phase is constantly occurring because of spindle speed variation and circuit parameter variations.

The operating specifications for this circuit are as follows:

Free Running Frequency	2MHz
Capture Range	$\pm 15\%$
Lock Up Time	50 microsec. "1111" or "0000" Pattern
	100 Microsec "1010" Pattern

The RAW READ pulses are generated from the falling edge of the SHAPED DATA pulses. The pulses are also reshaped to meet the 179X requirements. $VFO\ CLK \div 2$ OR 4 is divided by 2 once again to obtain $VFO\ CLK\ OUT$ whose frequency is that required by the 179X RCLK input. RCLK must be controlled by VFOE so VFOE is sampled on each rising edge of $VFO\ CLK\ OUT$. When VFOE goes active EN RCLK goes active in synchronization with $VFO\ CLK\ OUT$ preventing any glitches on the RCLK output. When VFOE goes inactive EN RCLK goes inactive in synchronization with $VFO\ CLK\ OUT$, again preventing any glitches on the RCLK output.

Figure 14 illustrates a PPL data recovery circuit using the Western Digital 1691 Floppy Support device. Both data recovery and Write Precomp Logic is contained within the 1691, allowing low chip count and PLL reliability. The 74S124 supplies the free-running VCO output. The PUMP UP and PUMP DOWN signals from the 1691 are used to control the 74S124's frequency.

COMMAND USAGE

Whenever a command is successfully or unsuccessfully completed, the busy bit of the status register is reset and the INTRQ line is forced high. Command termination may be detected either way. The INTRQ can be tied to the host processor's interrupt with an appropriate service routine to terminate commands. The busy bit may be monitored with a user program and will achieve the same results through software. Performing both an INTRQ and a busy bit check is not recommended because a read of the status register to determine the condition of the busy bit will reset the INTRQ line. This can cause an INTRQ from not occurring.

RESTORE COMMAND

On some disk drives, it is possible to position the R/W head outward past Track 00 and prevent the TROO line from going low unless a STEP IN is first performed. If this condition exists in the drive used, the RESTORE command will never detect a TROO. Issuing several STEP IN pulses before a RESTORE command will remedy this situation. The RESTORE and all other Type I commands will execute even though the READY bit indicates the drive is not ready (NOT READY = 1).

READ TRACK COMMAND

The READ TRACK command can be used to manually inspect data on a hard copy printout. Gaps, address marks and all data are brought in to the data register during this command. The READ TRACK command may be used to inspect diskettes for valid formatting and data fields as well as address marks. Since the 179X does not synchronize clock and data until the Index Address Mark is detected, data previous to this ID mark will not be valid. READ GATE (RG) is not actuated during this command.

READ ADDRESS COMMAND

In systems that use either multiple drives or sides, the read address command can be used to tell the host processor which drive or side is selected. The current position of the R/W head is also denoted in the six bytes of data that are sent to the computer.

TRACK	SIDE	SECTOR	CRS LENGTH	CRC 1	CRC 2
-------	------	--------	---------------	----------	----------

The READ ADDRESS command as well as all other Type II and Type III commands will not execute if the READY line is inactive (READY = 0). Instead, an interrupt will be generated and the NOT READY status bit will be set to a 1.

FORCED INTERRUPT COMMAND

The Forced Interrupt command is generally used to terminate a multiple sector command or to insure Type I status in the status register. The lower four bits of the command determine the conditional interrupt as follows:

- | | |
|-------|---------------------------------|
| 1_0 | = NOT-READY TO READY TRANSITION |
| 1_1 | = READY TO NOT-READY TRANSITION |
| 1_2 | = EVERY INDEX PULSE |
| 1_3 | = IMMEDIATE INTERRUPT |

Regardless of the conditional interrupt set, any command that is currently being executed when the Forced Interrupt command is loaded will immediately be terminated and the busy bit will be reset indicating an idle condition.

Then, when the condition for interrupt is met, the INTRQ line will go high signifying that the condition specified has occurred.

The conditional interrupt is enabled when the corresponding bit positions of the command ($1_3 - 1_0$) are set to a 1. If $1_3 - 1_0$ are all set to zero, no interrupt will occur, but any command presently under execution will be immediately terminated upon receipt of the Force Interrupt command (HEX DO).

As usual, to clear the interrupt a read of the status register or a write to the command register is required. The exception is when using the immediate interrupt condition ($1_3 = 1$). If this command is loaded into the command register, an interrupt will be immediately generated and the current command terminated. Reading the status or writing to the command register will not automatically clear the interrupt; another forced interrupt command with $1_3 - 1_0 = 0$ must be loaded into the command register in order to reset the INTRQ from this condition.

More than one condition may be set at a time. If for example, the READY TO NOT-READY condition ($1_1 = 1$) and the Every Index Pulse ($1_2 = 1$) are both set, the resultant command would be HEX "DA". The "OR" function is performed so that either a READY TO NOT-READY or the next Index Pulse will cause an interrupt condition.

DATA RECOVERY

Occasionally, the R/W head of the disk drive may get "off track", and dust or dirt may get trapped on the media. Both of these conditions will cause a RECORD NOT FOUND and/or a CRC error to occur. This "soft error" can usually be recovered by the following procedure:

1. Issue the command again
2. Unload and load the head and repeat step
3. Issue a restore, seek the track, and repeat step 1

If RNF or CRC errors are still occurring after trying these methods, a "hard error" may exist. This is usually caused by improper disk handling, exposure to high magnetic fields, etc. and generally results in destroying portions or tracks of the diskette.

FD179X

FIGURE 1. DEVICE CHARACTERISTICS

DEVICE	SNGL DENSITY	DBLE DENSITY	INVERTED BUS	TRUE BUS	DOUBLE-SIDED
1791	X	X	X		
1792	X		X		
1793	X	X		X	
1794	X			X	
1795	X	X	X		X
1797	X	X		X	X

FIGURE 2. STORAGE CAPACITIES *40 tracks*

SIZE	DENSITY	SIDES	UNFORMATTED CAPACITY (NOMINAL)		BYTE TRANSFER TIME	FORMATTED CAPACITY	
			PER TRACK	PER DISK		PER TRACK	PER DISK
5¼"	SINGLE	1	3125	109,375*	64µs	2304**	80,640
5¼"	DOUBLE	1	6250	218,750	32µs	4608***	161,280
5¼"	SINGLE	2	3125	218,750	64µs	2304	161,280
5¼"	DOUBLE	2	6250	437,500	32µs	4608	322,560
8"	SINGLE	1	5208	401,016	32µs	3328	256,256
8"	DOUBLE	1	10,416	802,032	16µs	6656	512,512
8"	SINGLE	2	5208	802,032	32µs	3328	512,512
8"	DOUBLE	2	10,416	1,604,064	16µs	6656	1,025,024

*Based on 35 Tracks/Side

**Based on 18 Sectors/Track (128 byte/sec)

***Based on 18 Sectors/Track (256 bytes/sec)

FIGURE 3. NOMINAL VS. WORSE CASE SERVICE TIME

SIZE	DENSITY	NOMINAL TRANSFER* TIME	WORST-CASE 179X SERVICE TIME	
			READ	WRITE
5¼"	SINGLE	64µs	55.0µs	47.0µs
5¼"	DOUBLE	32µs	27.5µs	23.5µs
8"	SINGLE	32µs	27.5µs	23.5µs
8"	DOUBLE	16µs	13.5µs	11.5µs

FIGURE 4A. FM RECORDING

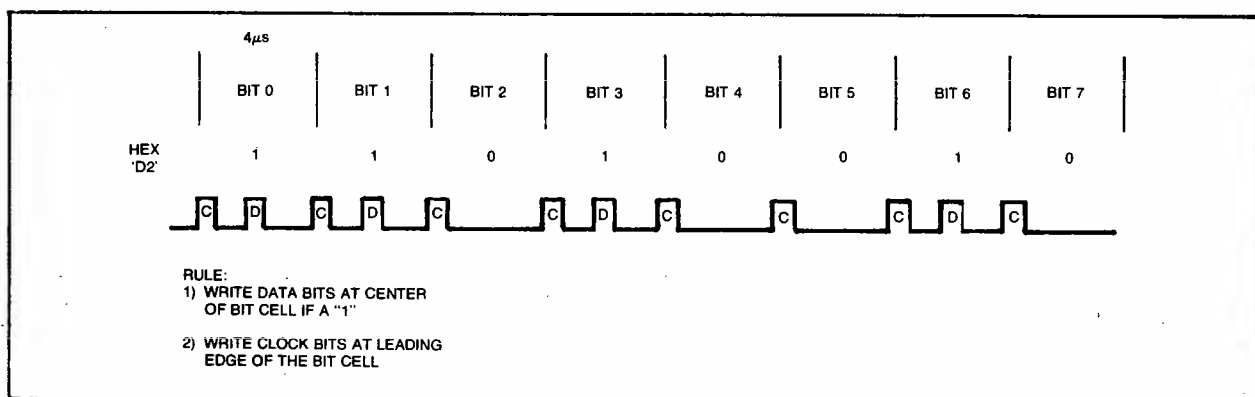
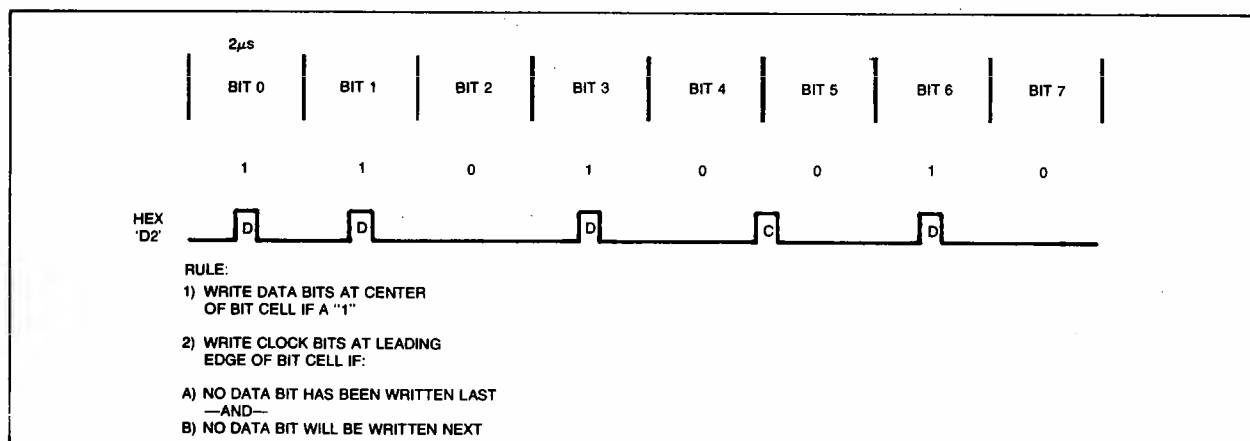


FIGURE 4B. MFM RECORDING



FD179X

FIGURE 5. WF/VFOE DEMULTIPLEXING CIRCUITRY

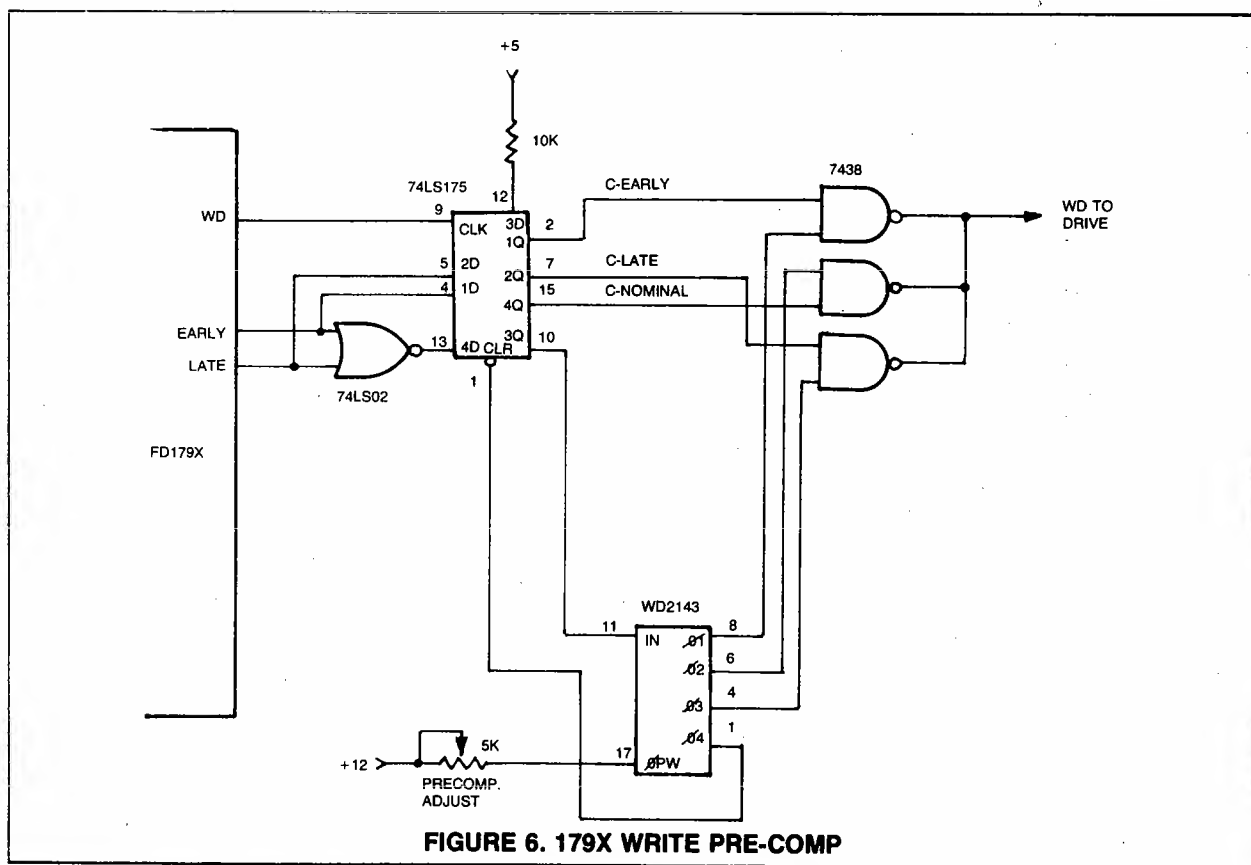
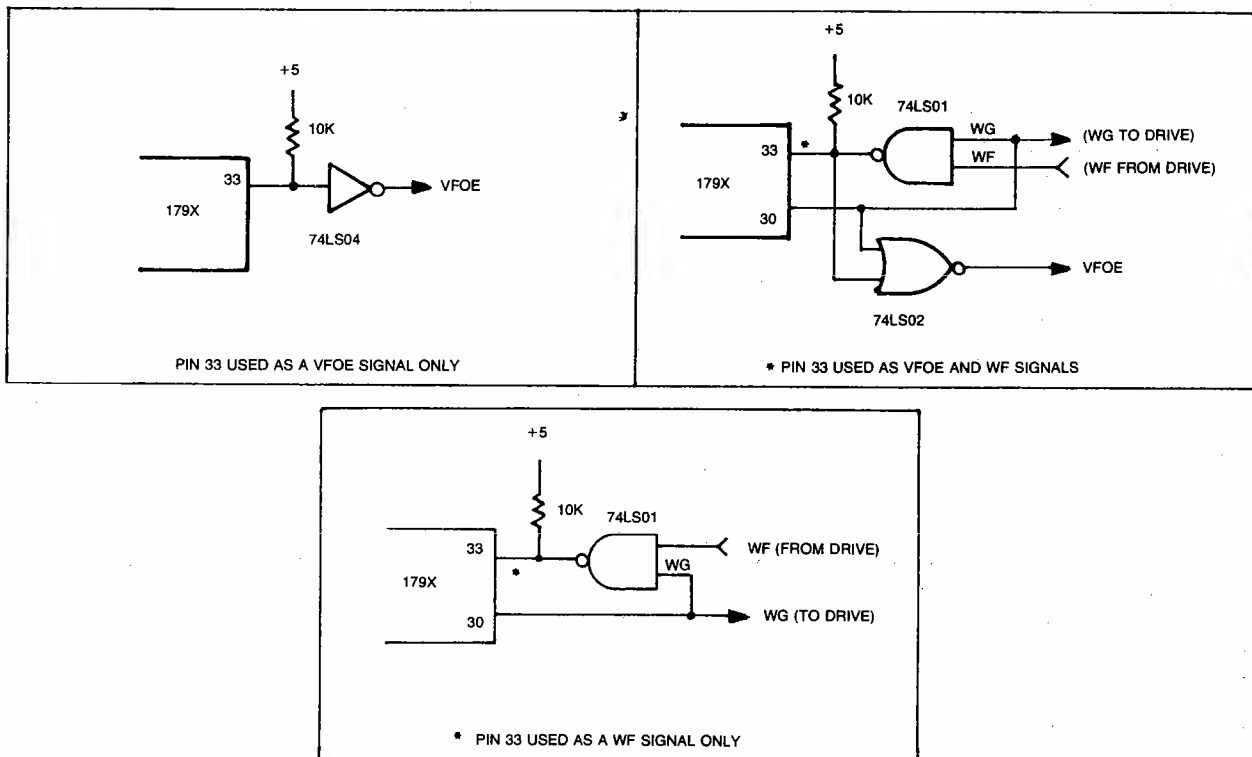


FIGURE 6. 179X WRITE PRE-COMP

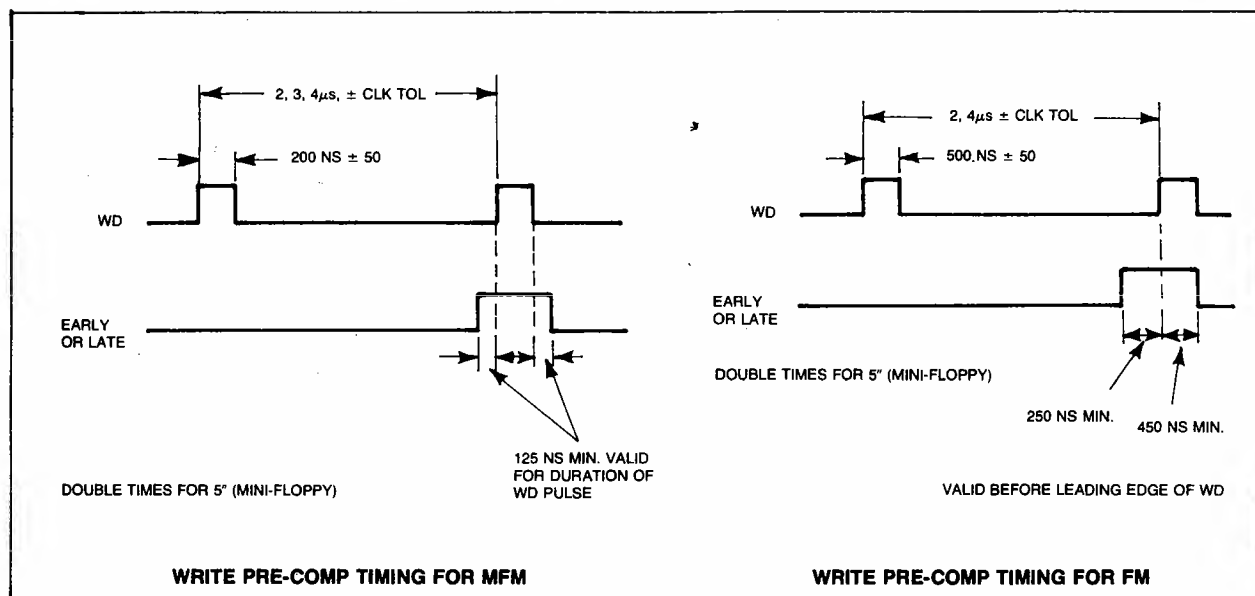


FIGURE 7. WRITE PRE-COMP TIMING

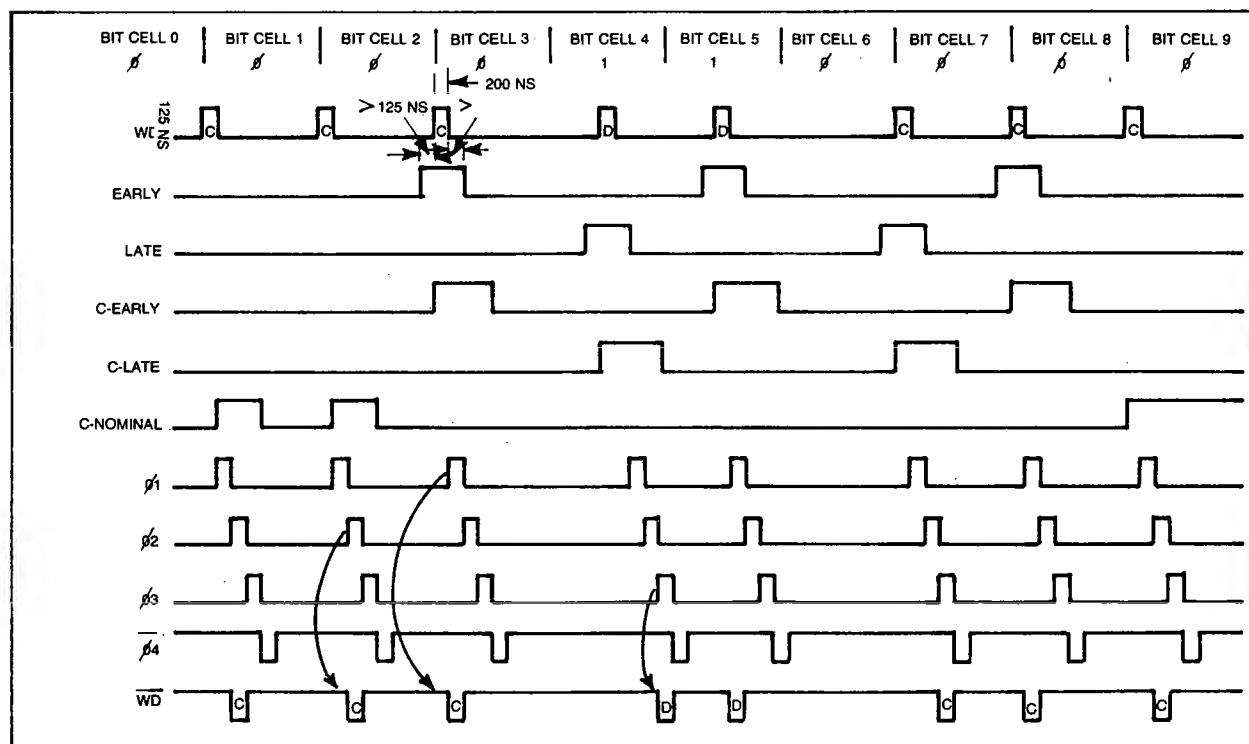


FIGURE 8. PRECOMP TIMING FOR CIRCUIT IN FIGURE 6

FD179X

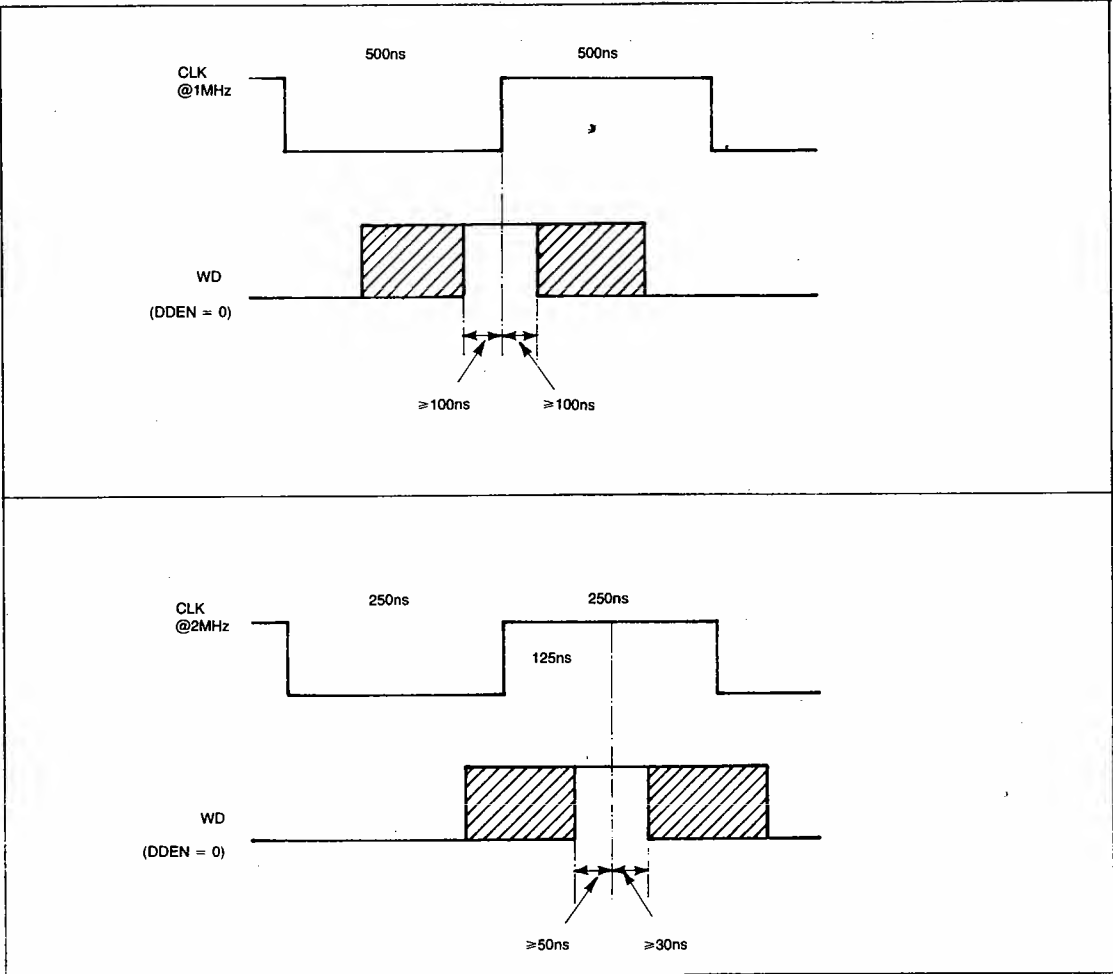


FIGURE 9. WD/CLK RELATIONSHIP FOR WRITE PRECOMP USE

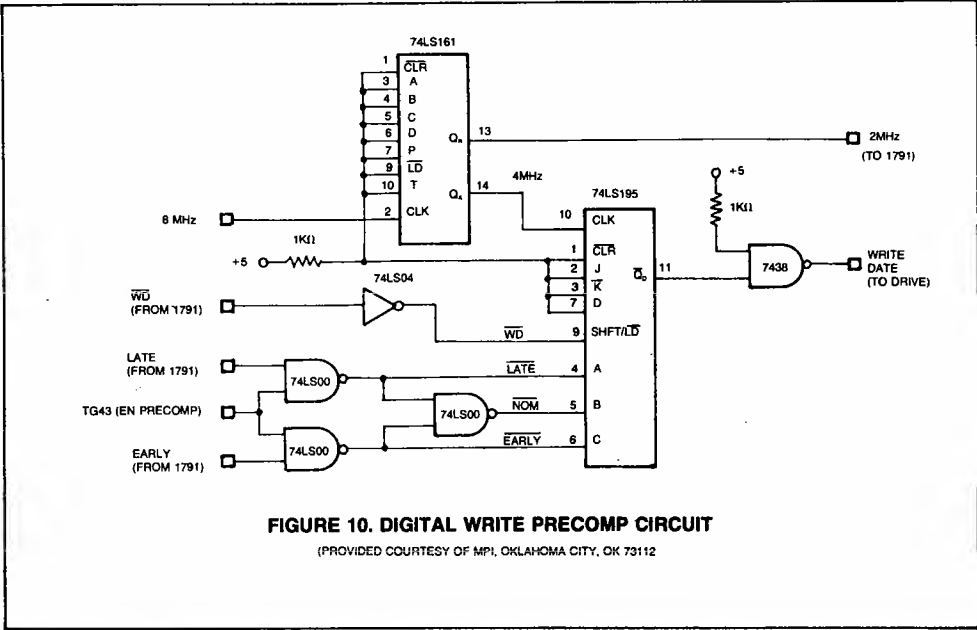


FIGURE 10. DIGITAL WRITE PRECOMP CIRCUIT
(PROVIDED COURTESY OF MPI, OKLAHOMA CITY, OK 73112)

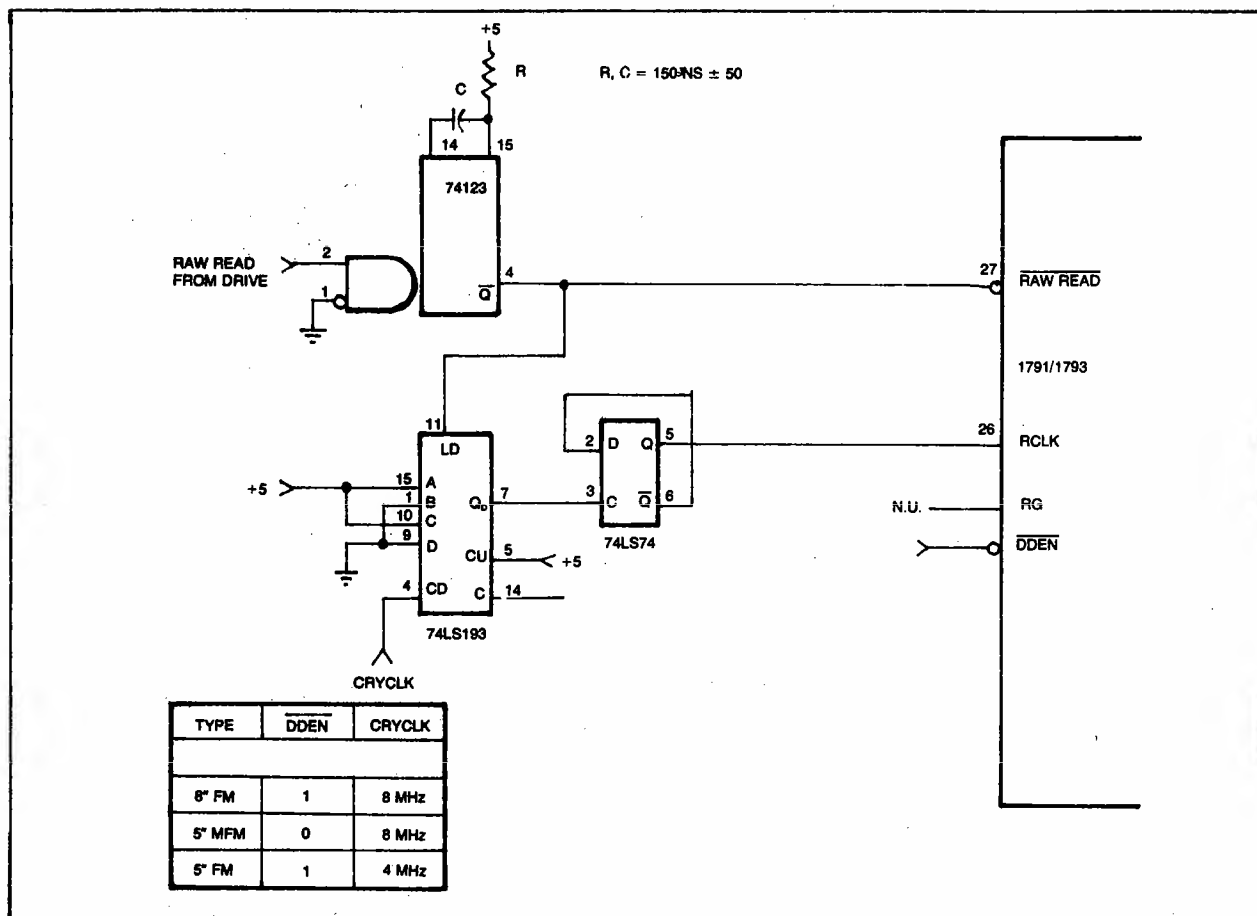


FIGURE 11. COUNTER/SEPARATOR

FD179X

745288 PROGRAMMING TABLE

ADDRESS	DATA	ACTION TAKEN
00	01	NONE
01	01	RETARD BY 1 COUNT
02	02	
03	03	
04	03	RETARD BY 2 COUNTS
05	04	
06	05	
07	06	
08	08	ADVANCE BY 2 COUNTS
09	0D	
0A	0C	
0B	0E	
0C	0F	
0D	0F	ADVANCE BY 1 COUNT
0E	00	
0F	01	
10	01	FREE RUN
11	02	
12	03	
13	04	
14	05	
15	06	
16	07	
17	07	
18	09	
19	0A	
1A	0B	
1B	0C	
1C	0D	
1D	0E	
1E	0F	
1F	00	

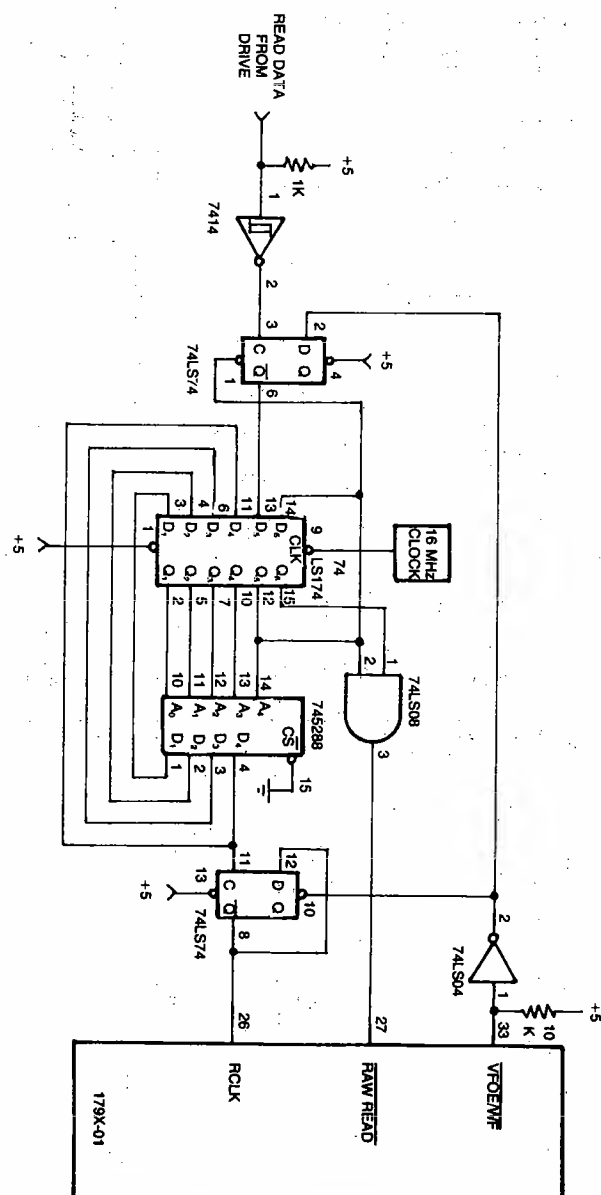


FIGURE 12. 179X DATA SEPARATOR

(PROVIDED COURTESY OF ANDROMEDA SYSTEMS, PANORAMA CITY, CA 91402)

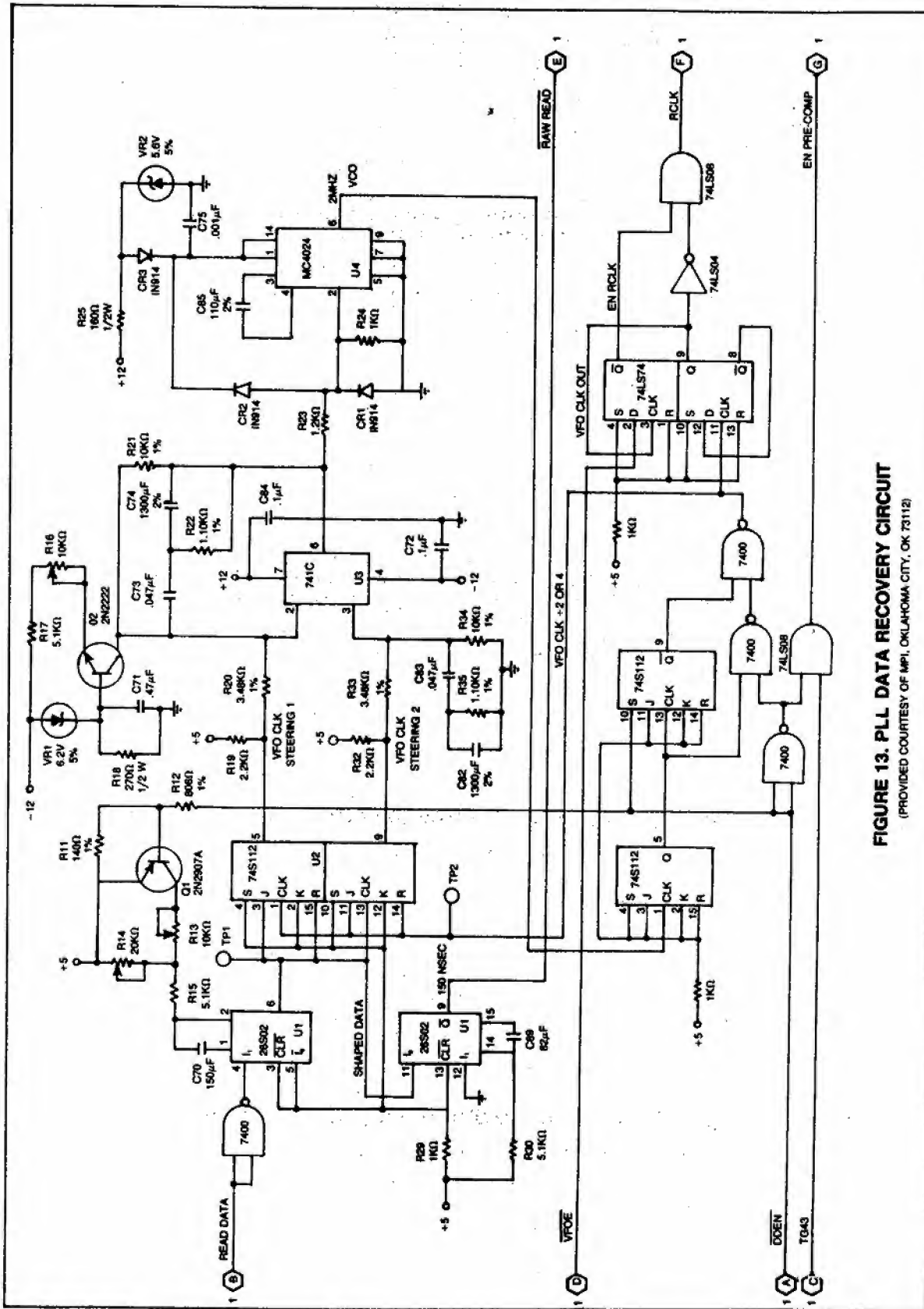


FIGURE 13. PLL DATA RECOVERY CIRCUIT
(PROVIDED COURTESY OF MP1, OKLAHOMA CITY, OK 73112)

FD179X

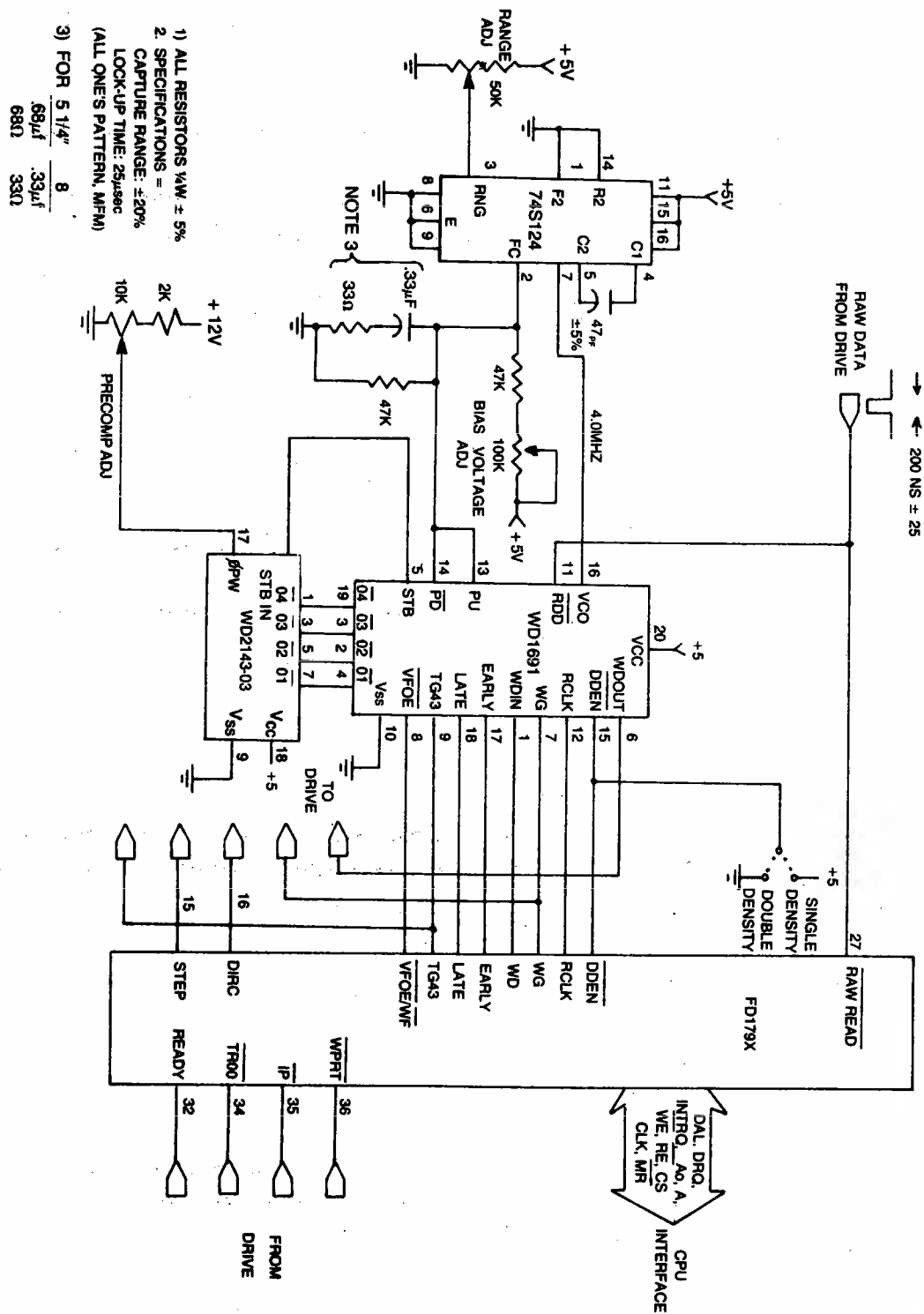


FIGURE 14. 8" SINGLE/DOUBLE DENSITY SYSTEM

Refer to 179X-02 Floppy Disk Formatter/Controller
Family Data Sheet for Command, Timing and Status
Information.

See page 725 for ordering information.

FD179X

FD179X

Information furnished by Western Digital Corporation is believed to be accurate and reliable. However, no responsibility is assumed by Western Digital Corporation for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Western Digital Corporation. Western Digital Corporation reserves the right to change specifications at anytime without notice.

Glossary

access

The operation of seeking, reading or writing data on a storage unit (in this case, the diskette).

access time

The time that elapses between any instruction being given to access some data and that data becoming available for use.

address

An identification (number, name, or label) for a location in which data is stored.

alphanumeric (characters)

A generic term for numeric digits and alphabetic characters.

assembly language

A machine-oriented language for programming mnemonics and machine readable code from the mnemonics.

base 2

The 'binary' numbering system consisting of more than one symbol, representing a sum, in which the individual quantity represented by each figure is based on a multiple of 2.

base 10

The 'decimal' numbering system — consisting of more than one symbol, representing a sum, in which the individual quantity represented by each symbol is based on a multiple of 10.

base 16

The 'hexadecimal' numbering system — consisting of more than one symbol representing a sum, in which the individual quantity represented by each symbol is based on a multiple of 16.

bit

A single 'binary' digit whose value is 'zero' or 'one'.

buffer

A small area of memory used for the temporary storage of data to be processed.

byte

Eight 'bits'. A 'byte' may represent any numerical value between '0' and '255'.

command file

A file consisting of a list of commands, to be executed in sequence.

control code

In programming, instructions which determine conditional jumps are often referred to as control instructions and the time sequence of execution of instructions is called the flow of control.

CRC error

Cyclic Redundancy Check. A means of checking for errors by using redundant information used primarily to check disk I/O while verifying

data type

The form in which data is stored; i.e., integer, single precision, double precision, 'alphanumeric' character strings or 'strings'.

DEC

Initials for Directory Entry Code.

device

Any physical or logical contrivance capable of accepting, processing or supplying data.

direct access

Retrieval or storage of data by a reference to its location on a disk, rather than relative to the previously retrieved or stored data.

DIRECT STATEMENT (IN FILE)

A program statement that exists in the disk file that is not assigned a line number.

DIRECTORY

A table giving the relationships between items of data. Sometimes a table or an index giving the addresses of data.

disk

The small magnetic disk used to store logical data.

disk drive

The mechanical device that rotates, reads and writes to a diskette.

displacement

A specified number of sectors, at the top or beginning of the file, in which the 'bookkeeping' and file parameters are stored for later use by various program modules.

driver

A routine that makes a particular device (usually hardware) work in a desired way.

DUMP

To transfer all or part of the contents of one section of computer memory or disk into another section, or to some other computer device.

EOF

Initials for 'end of file'. It is common practice to say that the EOF is record number nn or that the EOF is byte 15 of sector 12. Hence, it is a convenient term to use in describing the location of the last record or last byte in a file.

extent

A contiguous area of data storage.

file

A collection of related records treated as a unit; The word file is used in the general sense to mean any collection of informational items similar to one another in purpose, form and content.

file parameters

The data that describes or defines the structure of the file.

FILESPEC

A file specification and may include the 'file name', the 'the file name extension', 'password', and 'disk drive' specification.

file area

The physical location of the file, on the disk, or in memory.

header record

A record containing common, constant or identifying information for a group of records which follow.

integer

A natural or whole number with no decimal point.

key

A data item used to identify or locate a record or other data grouping.

label

A set of symbols used to identify or describe an item, record, message or file. Occasionally, it may be the same as the address in storage.

least significant byte

The significant byte contributing the smallest quantity to the value of a numeral.

load module

A program developed for loading into storage and being executed when control is passed to the program.

logical

An adjective describing the form of data organization, hardware or system that is perceived by an application program, programmer, or user; it may be different than the real (physical) form.

logical file

A file as perceived by an application program; it may be in a completely different form from that in which it is stored on the storage units.

logical operator

A mathematical symbol that represents a mathematical process to be performed on an associated operand. Such operators are 'AND', 'OR', 'NOT', 'AND NOT' and 'OR NOT'.

logical record

A record or data item as perceived by an application program; it may be in a completely different form from that in which it is stored on the storage units.

machine-language

See assembly language.

monitor

A program that may supervise the operation of another program for operation or debugging or other purposes.

most significant byte

The significant byte contributing the greatest quantity to the value of a numeral.

nibble

The four right most or left most binary digits of a byte.

null

An absence of information as contrasted with zero or blank for the presence of no information.

on-line

An on-line system is one in which the input data enter the computer directly from their point of origin, and/or output data are transmitted directly to where they are used. The intermediate stages such as writing tape, loading disks or off-line printing are avoided.

operating system

Software which enables a computer to supervise its own operations, automatically calling in programs, routines, language and data as needed for continuous throughput of different types of jobs.

parity

Parity relates to the maintenance of a sameness of level or count, i.e., keeping the same number of binary ones in a computer word to thus be able to perform a check based on an even or odd number for all words under examination.

physical record

A collection of bits that are physically recorded on the storage medium and which are read or written by one machine input/output instruction.

pointer

The address or a record (or other data groupings) contained in another record so that a program may access the former record when it has retrieved the latter record. The address can be absolute, relative, symbolic, hence, the pointer is referred to as absolute, relative, or symbolic.

primary entry

The main entry made to the directory.

random access

To obtain data directly from any storage location regardless of its position, with respect to the previously referenced information. Also called 'direct access'.

random access storage

A storage technique in which the time required to obtain information is independent of the location of the information most recently obtained.

read

To accept or copy information or data from input devices or a memory register; i.e., to read out, to read in.

record

A group of related fields of information treated as a unit by an application program.

relational operator

A mathematical symbol that represents a mathematical process to perform a comparison describing the relationship between two values (e.g. < *less than* . . . > *greater than* . . . *equal* . . . and combinations thereof).

search

To examine a series of items for any that have a desired property or properties.

sector

The smallest addressable portion of storage on a diskette.

seek

To position the access mechanism of a direct-access storage device at a specified location.

sequential access

Access in which records must be read serially or sequentially one after the other; i.e., ASCII files, tape.

sort

To arrange a file or data in a sequence by a specified key (may be alphabetic or numeric and in descending or ascending order).

source code

The text from which executable code is derived.

system file

A program used by the operating system to manage the executing program and/or the computer's resources.

vector

A line representing the properties of magnitude and direction. Since such a 'line' can be described in mathematical terms, a mathematical description (expressed in numbers, of course) of a given 'direction' and 'magnitude' is referred to as a 'vector'.

verify

To check a data transfer or transcription.

working storage

A portion of storage, usually computer main memory, reserved for the temporary results of operations.

write

To record information on a storage device.

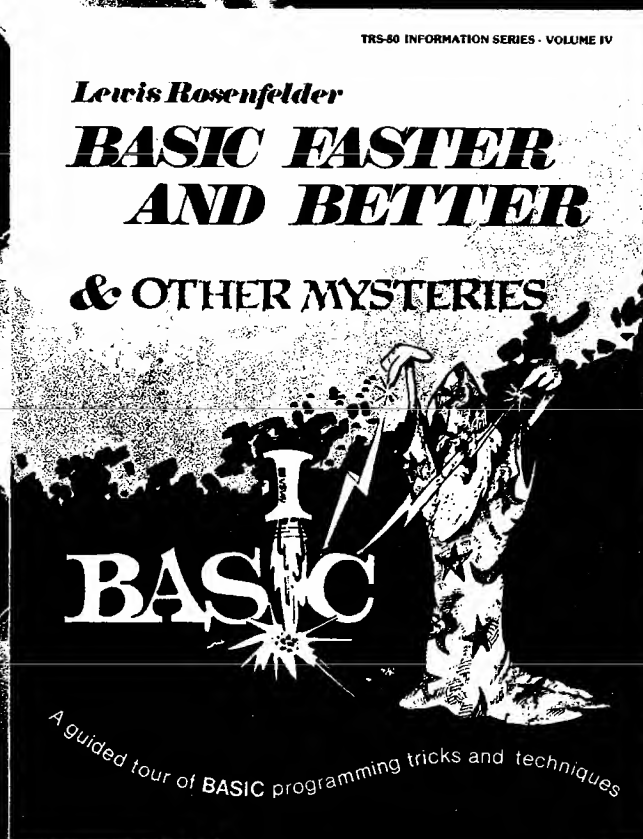
zap

To change a byte or bytes of data in memory on on diskette by using a software utility program.

notes

notes

MAKE BASIC PERFORM LIKE A CHAMPION



BASIC is not nearly as slow as most programmers think it is. *BASIC Faster and Better* shows you how to supercharge your BASIC, with 300 pages of fast, functions and subroutines.

You won't find any trivial 'check-book-balancing' programs in this book – it's packed with *useful* programs.

Tutorial for the beginner, instructive for the advanced, and invaluable

for the professional, this book doesn't just talk . . . it shows how! All of the routines are



1953 West
11th Street
Upland, CA
91786 (714)
946-5805

also available on disk, so that you can save hours of keyboarding and debugging.

BASIC Faster & Better and Other Mysteries is \$29.95, and the two program disks are \$19.95 each. The #1 disk *BFBDEM* contains all the demonstration programs, and #2 *BFB LIB* has all the library functions.

Get the book and/or disks from your local *JG* dealer and B. Dalton bookstores.



Computer Books & Software



Microsoft trademark Microsoft Corporation
Apple trademark Apple Computer Inc.
TRS-80 trademark TANDY Corporation
Electric Pencil ©1981 Michael Shrayner

BOOKS

TRS-80 Disk & Other Mysteries. H.C. Pennington.
The "How to" book of Data Recovery. 128 pages. **\$22.50**

Microsoft Basic Decoded & Other Mysteries.
James Farvour. The Complete Guide to Level II
Operating Systems & BASIC. 312 pages **\$29.95**

The Custom TRS-80 & Other Mysteries.
Dennis Bathory Kitsz. The Complete Guide to
Customizing TRS-80 Software & Hardware.
336 pages **\$29.95**

BASIC Faster & Better & Other Mysteries.
Lewis Rosenfelder. The Complete Guide to BASIC
Programming Tricks & Techniques. 290 pages **\$29.95**

Electric Pencil Operators Manual. Michael Shrayner.
Electric Pencil Word Processing System Manual.
123 pages. **\$24.95**

The Custom Apple. Winfried Hofacker & Ekkehard
Floegel. The Complete Guide to Customizing the Apple
Software and Hardware. 190 pages **\$24.95**

Machine Language Disk I/O & Other Mysteries.
Michael D. Wagner. A Guide to Machine Language Disk I/O
for the TRS-80 Models I and II.
Available October 1982 **\$29.95**

TRSDOS 2.3 Decoded & Other Mysteries.
James Lee Farvour. Commented Guide to TRSDOS 2.3 for
the Model I. Available December 1982 **\$29.95**

SOFTWARE

Electric Pencil. Michael Shrayner.
Word Processing System. Available in DISK **\$89.95**
STRINGY FLOPPY or CASSETTE **\$79.95**

Red Pencil. Automatic Spelling Correction Program.
For use with the Electric Pencil Word Processing System.
Available in DISK ONLY **\$89.95**

Blue Pencil.* Dictionary - Proofing Program.
For use with the Electric Pencil Word Processing System.
Available in DISK ONLY **\$89.95**

* Requires Red Pencil program.

BFBLIB. Lewis Rosenfelder. Basic Faster & Better
Library Disk. 32 Demonstration Programs. Basic Overlays.
Video Handlers. Sorts & more for the Model I & II.
Available in DISK ONLY **\$19.95**

BFBDEM. Lewis Rosenfelder. Basic Faster & Better
Demonstration Disk. 121 Functions, Subroutines &
User Routines for the TRS-80 Model I & II.
Available in DISK ONLY **\$19.95**

Prices Subject to change without notice

Add \$4.00 shipping and handling charge per item.

California residents add 6% sales tax. Canadian residents add 20% for exchange rate.



1953 West
11th Street
Upland, CA
91786 (714)
946-5805



ISBN 0-936200-06-5